Customizing NFuse™ Classic1.7

NFuse Classic 1.7

April 2002

Citrix Systems, Inc.

Last Updated: April 3, 2002 6:35 (LBL)

Contents

Chapter 1 Welcome to Citrix NFuse Classic 7
Who Should Read this Guide 7
The NFuse Classic Programming Interface 8
Overview of This Manual 8
Document Conventions 9
Finding More Information 10
Citrix Developer Network 11
Citrix on the World Wide Web 11
Providing Feedback About This Guide 12
Disclaimer 12

Chapter 2 Using Citrix SubstitutionTags 13
NFuse Classic Substitution Tags 13
Simple Web Site Creation 14
Template ICA File Creation 15
Generating ICA files 17
Session Fields 17
Substitution Tag and Session Field Reference 18
General Tags 18
Application Property Tags 25
User Interface Tags 30
Conditional Tags 31
Citrix Secure Gateway 33

Chapter 3 Java Object Reference 35
NFuse Classic Java Objects 36
CitrixWireGateway 36
ClearTextCredentials 42
GroupCredentials 44
AppEnumerator 47
App 53
AppSettings 74
AppDataList 79
AppListCache 81
TemplateParser 84
PropertiesKeys 90
AnonCredentials 92
AuthenticatedCredentials 92

Chapter 4 ICA File Reference 95
ICA File Structure 96
[WFClient] 96
[ApplicationServers] 97
[ApplicationName] 97
ICA File Parameters 97
General Parameters 98
User Credential Parameters 100
Window Size and Color Parameters 101
Persistent Caching Parameters 102
TCP/IP Browsing Parameters 103
Setting NFuse Classic Server Location Options 104
Setting Encryption Parameters 106
Configuring Authentication Over Encrypted Connections 107
Setting SOCKS Proxy Parameters 109
SpeedScreen Latency Reduction Parameters 111
Client Auto Update Parameters 111
Client Device Mapping Parameters 112

Chapter 5 Example Scripts 113
Introduction 113
How to Use the Scripts 114
ASP Examples 115
Simple (Flat) Application Iteration 115
Text List of Application Names 115

Text List of Names, Details and Icons 116
Tabular Listing of Applications, Details, and Icons 117
Folder Iteration 118
Text Listing of Root Folders 118
Clickable Folder Navigation 119
Clickable Folder Navigation, and Application Names 120
Launching Applications 121
Launchable Application Lists 121
Seamless Windows 124
Embedding Applications 127
Disabling Ticketing 132
Passing User Credentials/Login 134
Caching 138
JSP Examples 142
Simple (Flat) Application Iteration 142
Text List of Application Names 142
Text List of Names, Details and Icons 143
Tabular Listing of Applications, Details, and Icons 144
Folder Iteration 146
Text Listing of Root Folders 146
Clickable Folder Navigation 148
Clickable Folder Navigation, and Application Names 149
Launching Applications 152
Launchable Application Lists 152
Seamless Windows 156
Embedding Applications 160
Disabling Ticketing 166
Passing User Credentials/Login 171
Caching 177

CHAPTER 1

Welcome to Citrix NFuse Classic

Welcome to NFuse Classic, Citrix's application management and deployment system. NFuse Classic combines the centralized application management capabilities of MetaFrame server software with new techniques for Web application deployment into a customizable application delivery mechanism.

This is a Web master's application, placing complete control over the application deployment process in the hands of the administrator. Using NFuse Classic, you can create standalone Web sites for application access or Web sites that can be integrated into your corporate portal.

NFuse Classic brings a powerful user interface to the application deployment process. This interface uses Java object technology executed on a Web server to dynamically create an HTML-based presentation of the MetaFrame server farm for each of your users. Included in each user's presentation are all of the applications published in the MetaFrame server farm for that user.

Who Should Read this Guide

This guide is for MetaFrame server administrators and Web masters who want to extend and customize the capabilities of NFuse Classic1.7 within your organization. This document supplements the *NFuse Classic Administrator's Guide*.

Topics covered include:

- Using NFuse Classic tags
- NFuse Classic Java Object reference
- ICA File reference
- Example scripts and code that can be cut and pasted into your HTML pages

The NFuse Classic Programming Interface

NFuse Classic's Java objects provide functionality that you can access using its application programming interface. This programming interface lets you customize your own sites according to the requirements of your environment.

Web masters can use the following to access the programming interface:

- **Microsoft's Active Server Pages**. Using Active Server Pages, Web masters can write Web server scripts that implement NFuse Classic Java object functionality. Such scripts call the Java objects to perform various functions and then place the results of those functions in plain HTML documents or ICA files.
- **Sun Microsystems' JavaServer Pages**. Like Active Server Pages, JavaServer Pages provide Web masters with a Web server scripting environment. JavaServer Pages support makes NFuse Classic compatible with many Java Web servers.
- **Citrix substitution tags**. Citrix substitution tags provide Web masters who are unfamiliar with Web server scripting a simplified method of accessing the NFuse Classic Java objects. Substitution tags are proprietary HTML extensions that Web masters can write into plain HTML documents to create simple Web sites. Advanced users can create substitution-tag-based sites that use a Java servlet or Active Server Page support files to perform necessary tasks.

In addition to the methods listed above, you can also write your own Java servlets using the NFuse Classic Java objects.

Overview of This Manual

This manual contains the following chapters:

Chapter 2: "Using Citrix subsitution Tags"
: Lists Citrix substitution tags and explains their usage. This chapter includes a tutorial that describes how to make modifications to substitution-tag-based Web sites.

Chapter 3: "Java Object Reference"
: Explains the NFuse Classic application programming interface. This chapter lists objects and methods, giving examples of how to access them from Active Server Page and JavaServer Page-based Web documents.

Chapter 4: "ICA File Reference"
: Explains the ICA file format and lists ICA file parameters you can add to the template ICA files used by NFuse Classic.

Chapter 5: “Example Scripts”
Example code is provided to allow you to add advanced functionality such as server redundancy and performance enhancements.

Document Conventions

The following conventional terms, text formats, and symbols are used throughout the printed documentation:

Convention	Meaning
Bold	Indicates column headings, command-line commands and options, dialog box titles, lists, menu names, tabs, and menu commands.
Italic	Indicates a placeholder for information or parameters that you must provide. For example, if the procedure asks you to type a filename, you must type the actual name of a file. Italics also indicate new terms and the titles of other books.
ALL UPPERCASE	Represents keyboard keys (for example, CTRL, ENTER, F2).
...(ellipsis)	Indicates a command element can be repeated.
`Monospace`	Represents examples of screen text or entries that you might type at the command line or initialization files.
Code Sample	Example code appears in front of a gray background, as in the example below:

	`<html>`
	`<body></body>`
	`</html>`

➢	Indicates a procedure with sequential steps.
▪	Indicates a list of related information, not procedural steps.

Finding More Information

NFuse Classic includes the following documentation:

- The *Citrix NFuse Classic Administrator's Guide* tells administrators how to install and configure NFuse Classic, and can be found on the Components CD.
- The Readme file contains last minute updates, corrections to the documentation, and a list of known problems. This file is located on the Components CD within the NFuse folders.
- This document, *Customizing NFuse Classic*.

Other sources of information about NFuse Classic and other Citrix products:

- The Citrix *MetaFrame XP for Windows, Version 1.0, Feature Release 2 Administrator's Guide* explains how to install and configure MetaFrame XP Feature Release 2 on Windows servers. Included in this documentation is information about publishing applications, configuring the Citrix XML Service, and configuring the Citrix SSL Relay. The *MetaFrame XP Administrator's Guide* is on the MetaFrame XP CD-ROM.
- The *Feature Release 1 and Service Pack 3 Installation Guide for Citrix MetaFrame for Windows Version 1.8* tells administrators how to install and configure Service Pack 3 and Feature Release 1 on MetaFrame 1.8 for Windows servers. Included in this documentation is information about configuring the Citrix XML Service and the Citrix SSL Relay. The Installation Guide is available on the Feature Release 1/Service Pack 3 CD and on the Citrix download site.
- The *Citrix MetaFrame for UNIX Operating Systems, Feature Release 1 for Version 1.1, Administrator's Guide* tells administrators how to install and configure MetaFrame for UNIX. Included in this documentation is information about publishing applications and how to configure the XML Relay for UNIX. The *MetaFrame for UNIX Administrator's Guide* is available on the MetaFrame for UNIX, Feature Release 1 CD.
- The *Citrix SSL Relay for UNIX Administrator's Guide* is for system administrators who are responsible for installing, configuring, and maintaining Citrix SSL Relay on MetaFrame for UNIX Operating Systems servers. This guide is available on the *MetaFrame for UNIX Feature Release 1 CD-ROM* and on the Citrix Web site.
- The *Citrix Enterprise Services for NFuse Administrator's Guide Version 1.7* tells administrators how to install and configure Enterprise Services for NFuse. This guide is on the Components CD-ROM and on the Citrix Web site.
- The *Citrix Secure Gateway Administrator's Guide Version 1.0* tells administrators how to install and configure Citrix Secure Gateway. This guide is on the Components CD-ROM and on the Citrix Web site.

Citrix Developer Network

The Citrix Developer Network (CDN) is a Citrix program that extends the reach of Citrix application server technology to independent software vendors, independent hardware vendors, system integrators, ICA licensees, and corporate IT developers who want to incorporate MetaFrame server-based computing solutions into their products.

The Citrix Developer Network is a membership program with open enrollment. Through the new CDN Web site, Citrix provides access to developer tool kits, technical information, and test programs needed to successfully “design in” or add MetaFrame server-based computing compatibility to hardware and software. Today, the CDN program includes several software development kits (SDKs) and test kits, with an emphasis on delivering enabling technologies that promote technical relationships with Citrix.

Register for the Citrix Developer Network at the CDN Web site:

http://www.citrix.com/cdn

Citrix on the World Wide Web

The Citrix Web site, at http://www.citrix.com, offers a variety of information and services for Citrix customers and users. From the Citrix home page, you can access Citrix online Technical Support Services and other information designed to assist NFuse Classic administrators, including the following:

- Downloadable Citrix ICA Clients (available at http://www.citrix.com/download)
- Program information about Citrix Preferred Support Services options
- An FTP server containing the latest service packs, hotfixes, utilities, and product literature for download
- An online Solution Knowledgebase containing an extensive collection of technical articles, troubleshooting tips, and white papers
- Interactive online Solution Forums for discussion of technical issues with other users
- Frequently Asked Questions pages with answers to common technical and troubleshooting questions
- Citrix Documentation Library containing the latest MetaFrame documentation
- Information about programs and courseware for Citrix training and certifications
- Contact information for Citrix headquarters, including worldwide, European, Asia Pacific, and Japan headquarters

Providing Feedback About This Guide

It is our goal to provide you with accurate, clear, complete, and usable documentation for Citrix products. If you have any comments, corrections, or suggestions for improving our documentation, we would be happy to hear from you.

You can send e-mail to the documentation authors at documentation@citrix.com. Please include the product name and version number, and the title of the document in your message.

Disclaimer

The software (the "Software Development Kit") and accompanying documentation included or described in this document are provided to you, the licensed user, by Citrix Systems, Inc. ("Citrix") pursuant to the terms of the attached Software Development Kit License Agreement and may only be used and copied in accordance with the express terms of the Software Development Kit License Agreement. The licensed user may use the Software Development Kit and accompanying documentation solely to develop applications that access or utilize the Citrix Server Software. The licensed user may make one (1) copy of the Software Development Kit in machine-readable form solely for backup purposes, provided that the licensed user reproduces all proprietary notices on the copy. The licensed user may not reproduce, copy, modify, transfer or transmit by any means, electronic, mechanical or otherwise, the Software Development Kit or accompanying documentation, in whole or in part, or grant any rights in the Software Development Kit or accompanying documentation except as expressly provided in the Software Development Kit License Agreement.

The Software Development Kit, including all accompanying documentation contained herein, is provided "AS IS" and Citrix and its suppliers make and the licensed user receives no warranties or conditions, express, implied, statutory, or otherwise. Citrix has made reasonable efforts to ensure the completeness and accuracy of all information contained in this Software Development Kit.

The Software Development Kit and accompanying documentation are subject to change without notice and Citrix is not obligated to maintain, update, or otherwise support such Software Development Kit and accompanying documentation.

CHAPTER 2

Using Citrix SubstitutionTags

This chapter describes substitution tags and session fields.These substitution tags and session fields provide an interface to the NFuse Classic Java objects. You can use this interface to:

- Write HTML pages and ICA file templates that serve published applications to ICA Client users
- Modify the NFuse Classic configuration file (NFuse.conf) and, when used with HTTP cookies and URLs, create cross-page state in your Web sites

Note Some configuration file changes can be made from the Windows browser-based configuration tool that is a part of the NFuse Classic installation. See the *NFuse Classic Administrator's Guide* for more details.

This chapter includes a reference that lists substitution tag and session field syntax.

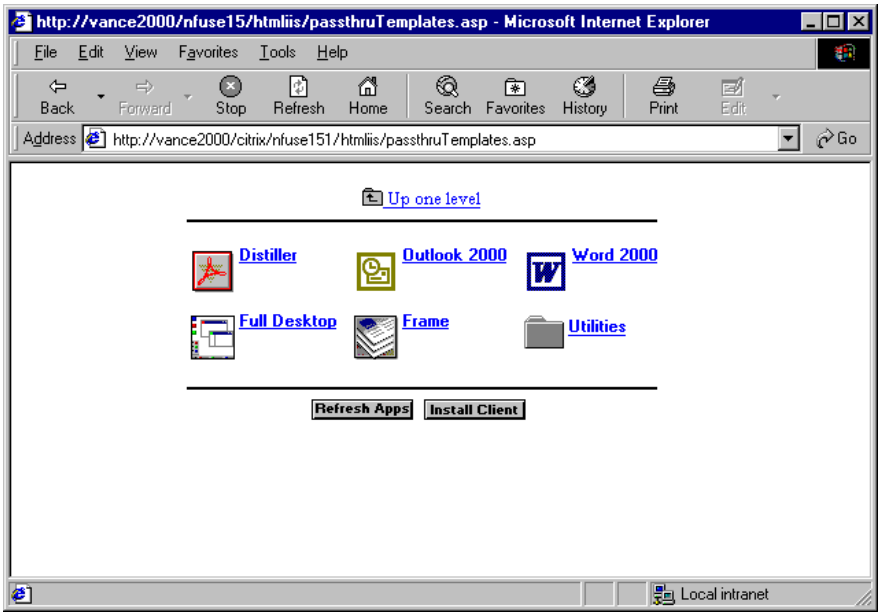
NFuse Classic Substitution Tags

The NFuse Classic substitution tags comprise a mark-up language that the Java objects can parse and deploy to Web browsers. Web masters can place substitution tags in template HTML and ICA files. When serving a template HTML or ICA file containing substitution tags, the Java object called TemplateParser scans the file for substitution tags and replaces them with relevant data. In the case of HTML pages, TemplateParser replaces the substitution tags with hyperlinks to published applications that the current user can access. When parsing a template ICA file, TemplateParser replaces the substitution tags with information that can be placed in an ICA file and sent to the client device's ICA Client for session initialization.

Simple Web Site Creation

NFuse Classic includes substitution tags for those Web masters who want to create customized Web pages but are unfamiliar with Web server scripting. Much like scripting-based Web pages, such as pages that make use of Microsoft's Active Server Pages or Sun's JavaServer Pages technologies, substitution-tag-based pages can dynamically present published application information to Web browsers on ICA Client devices.

Substitution tags are best used in simple pages. Unlike scripting-based pages, which give you the ability to precisely place and order published application links in a Web page, substitution-tag-based pages are intended for drawing published application links in a simple HTML table. The following picture illustrates a simple substitution-tag-based display of published application links:



Important For more advanced deployment scenarios, including Web page layouts that require exact placement and ordering of applications within a page, use scripting-based Web pages. See "Java Object Reference" on page 35.

When responding to a request from a Web browser for a substitution-tag-based Web page, the Java objects on the Web server perform symbol substitution on all the substitution tags contained in the page. This symbol substitution replaces the tags with user-specific application information and sends the browser a plain HTML page containing links to the applications in the user's application set.

To use substitution-tag-based Web sites with Microsoft Internet Information Server, you must have a Web server that supports Active Server Pages. Substitution-tag-based pages for Microsoft IIS use Active Server Page support files to perform the symbol substitution and various other NFuse-related tasks such as calling the Java objects to pass user credentials to the server farm and creating ICA files for the applications users select to run.

Substitution-tag-based pages for Apache, Netscape, and iPlanet Web servers use Java servlets to perform symbol substitution. To use substitution-tag-based Web sites with these servers, make sure the servers support Java servlets.

Template ICA File Creation

An *ICA file* is a text file containing information about a published application. ICA files are written in Ini file format and organize published application information in a standardized way that ICA Clients can interpret. When an ICA Client receives an ICA file, it initializes a session containing the desired application on the MetaFrame server specified in the file. All application links in NFuse Classic Web pages generate ICA files.

NFuse Classic uses a template ICA file as the basis for the ICA files referenced in its hyperlinks. A *template ICA file* is a text file that adheres to the ICA file format and contains substitution tags that can be replaced with information about a specific user and the desired application.

When a user clicks an NFuse Classic hyperlink, the Java objects retrieve the template ICA file and replace its substitution tags with information about the requested application and the user's credentials before sending the file to the client device.

The following is an example template ICA file:

```
<[NFuse_setSessionField NFuse_ContentType=application/x-ica]>
<[NFuse_setSessionField NFuse_WindowType=seamless]>

[WFClient]
Version=2
ClientName=[NFuse_ClientName]

[ApplicationServers]
[NFuse_AppName]=

[[NFuse_AppName]]
Address=[NFuse_IPV4Address]
InitialProgram=#[NFuse_AppName]
DesiredColor=[NFuse_WindowColors]
TransportDriver=TCP/IP
WinStationDriver=ICA 3.0
<[NFuse_IFSESSIONFIELD sessionfield="NFUSE_ENCRYPTIONLEVEL" value="basic"]>
Username=[NFuse_User]
Domain=[NFuse_Domain]
Password=[NFuse_PasswordScrambled]
<[/NFuse_IFSESSIONFIELD]>
<[NFuse_IFSESSIONFIELD sessionfield="NFUSE_SOUNDTYPE" value="basic"]>
ClientAudio=On
<[/NFuse_IFSESSIONFIELD]>
[NFuse_IcaWindow]

[NFuse_IcaEncryption]
```

Bracketed items such as [NFuse_AppName] and [NFuse_IPV4Address] indicate substitution tags that the TemplateParser object will replace when processing this template. Bracketed section headings such as [ApplicationServers] are not substitution tags; these bracketed items are standard ICA file section names and are always enclosed in brackets.

The presence of parameter/value pairs that do not contain substitution tags, such as the entry TransportDriver=TCP/IP. A template ICA file can contain any valid ICA file parameter/value pair specified along with the substitution-tag-based parameter/value pairs. For a list of valid ICA file entries, see “ICA File Reference” on page 95.

Generating ICA files

If you write web pages that generate ICA files, use your template or script page to set **sessionfield.NFuse_ContentType**. If you do not set this session field, you can explicitly define it as shown below in ASP pages:

```
Response.contentType="application/x-ica"
```

or

```
Response.setcontentType("application/x-ica")
```

If you set **sessionfield.NFuse_contentType** in any template ICA file or using scripts, a method on TemplateParser object **getContentType** will return the **contentType** set using the **sessionfield.NFuse_contentType**. This can later be used in the scripts.

Example

```
set parser = Server.CreateObject("com.citrix.nfuse.TemplateParser")
set contentType  = parser.getContentType() <!--- this will get the content type set via the
sessionfield.NFuse_contentType-->
```

Later in the code, add the following:

```
response.setContentType(contentType) <!-- contentType obtained via the TemplatePlate parser object
getContentType method-->
```

Session Fields

Session fields allow you to set NFuse Classic properties and, when used with HTTP cookies and URLs, to maintain state information between Web pages.

By using session fields in your Web pages, you can set and modify the MetaFrame server and TCP/IP port used for published application requests, the location of NFuse Classic's icon cache, and the directory where NFuse Classic stores template files, among other properties.

Session fields also allow you to maintain state information between NFuse ClassicWeb pages. State information can include the current user's password and user name or the properties of a published application, among others. You can use session fields to make these properties persist between pages so that they are available to the Java objects from different pages in your site.

For example, by placing user credential session fields in an HTTP cookie, you can make the login information entered by a user in a login form available to the Java objects parsing a template ICA file for a specific application. When the TemplateParser object performs symbol substitution on the template ICA file, it replaces user credential substitution tags in the template ICA file with the value of the user credential session fields specified in the cookie.

Note You can use session fields in substitution-tag-based pages and in scripting-based pages.

You can set session fields from various places in your Web site including in server script, URLs, cookies, and the NFuse.conf file. Session fields obey an override precedence order. This precedence order lets you set a session field to different values at different points in the execution of your site.

Substitution Tag and Session Field Reference

All substitution tags and session fields use the delimiters [...] or <[...]>. All substitution tags begin with *NFuse_*, such as <[NFuse_Domain]>. White space can appear inside the delimiters or between the delimiters and the name of the tag.

The following topics list NFuse Classic's substitution tags. Unless otherwise indicated, each tag functions as both a substitution tag and a session field. When processing a tag, the TemplateParser object replaces the tag with the current value of the corresponding session field. For example, when the TemplateParser object encounters the [NFuse_User] tag in a template ICA file, it replaces the tag with the current value of the [NFuse_User] session field. This session field must be set earlier in the execution of the site and made available to the template ICA file through a cookie or in a URL.

General Tags

These tags perform basic functions such as supplying the Java objects with MetaFrame server contact information and packaging of user credentials.

NFuse_Application

Specifies an instance of a published application.

NFuse_CitrixServer

Specifies which MetaFrame server provides the published application information. This value can be a fully-qualified DNS name, Windows NT server name, or IP address.

NFuse_CitrixServerPort

Specifies the TCP/IP port used by the MetaFrame server specified in **NFuse_CitrixServer** for NFuse Classic communication. The default value is the port number entered during Web Server Extension installation. This port number must match the port number used by the MetaFrame server.

NFuse_Transport

Specifies the protocol used to transport NFuse Classic data between the Web server and MetaFrame server specified in **NFuse_CitrixServe**r. If NFuse_Transport is not present, the connection uses HTTP.

Values

HTTP

Use HTTP to send the NFuse Classic data over a standard HTTP connection to the server and port specified in **NFuse_CitrixServer** and **NFuse_CitrixServerPort**.

SSL

Use SSL to send data over a secure connection that uses Citrix SSL Relay running on a MetaFrame server to perform host authentication and data encryption. This protocol sends the data to the server and port specified in **NFuse_CitrixServer** and **NFuse_CitrixServerPort**.

HTTPS

Use this option to send data over a secure HTTP connection using SSL, to the server specified in **NFuse_CitrixServer** and **NFuse_CitrixServerPort**.

NFuse_CurrentFolder

Determines what Program Neighborhood folder to enumerate when processing the **[NFuse_DrawPN]** and **[/NFuse_DrawPN]** tags. You can also use this tag to determine the value of the **[NFuse_SubFolder]** and **[NFuse_ParentFolder]** tags. If this session field is not set, the root folder is assumed.

NFuse_CurrentFolderUrlEncoded

Determines what Program Neighborhood folder to enumerate when processing the **[NFuse_DrawPN]** and **[/NFuse_DrawPN]** tags. You can also use this tag to determine the value of the **[NFuse_SubFolder]** and **[NFuse_ParentFolder]** tags. If this session field is not set, the root folder is assumed.

This tag returns data in a URL-encoded string. This format is useful when the returned string must be placed in a URL, cookie, or other location where certain characters, if left unencoded, can cause errors in a Web browser or Web server.

NFuse_SubFolder

This tag is replaced with the currently enumerated subfolder in a **[NFuse_DrawPN]**…**[/NFuse_DrawPN]** block.

NFuse_SubFolderUrlEncoded

This tag is replaced with the currently enumerated subfolder in a **[NFuse_DrawPN]**…**[/NFuse_DrawPN]** block.

This tag returns data in a URL-encoded string. This format is useful when the returned string must be placed in a URL, cookie, or other location where certain characters, if left unencoded, can cause errors in a Web browser or Web server.

NFuse_ParentFolder

This tag is replaced with the parent folder of the folder specified by the **NFuse_CurrentFolder** session field. If the **NFuse_CurrentFolder** session field is the root, this tag is replaced by the root folder (an empty string).

NFuse_ParentFolderUrlEncoded

This tag is replaced with the parent folder of the folder specified by the **NFuse_CurrentFolder** session field. If the **NFuse_CurrentFolder** session field is the root, this tag is replaced by the root folder (an empty string).

This tag returns data in a URL-encoded string. This format is useful when the returned string must be placed in a URL, cookie, or other location where certain characters, if left unencoded, can cause errors in a Web browser or Web server.

NFuse_Template

Specifies a template ICA or HTML file to parse. Specify a filename or sub-path located in the templates directory as specified by the **NFuse_TemplatesDir** session field.

NFuse_User

Specifies the client device user's user name.

NFuse_Domain

Specifies the client device user's domain.

NFuse_Password

Specifies the client device user's password.

NFuse_PasswordScrambled

Specifies an encoded form of the value of the **NFuse_Password** session field. The encoding used is expected by the ICA Client in ICA files. Using this tag without first setting **NFuse_Password** causes an error.

NFuse_Ticket

Retrieves authentication ticket for placement in ICA files. The TemplateParser object replaces this tag with all required credential information in the following format:

```
User=actual user name
Domain=\ (backslash) character followed by the last 16 characters of the ticket
ClearPassword=first 14 characters of ticket
```

Other information that may be retrieved includes the ICA file setting **UseLocalUserAndPassword**. This may be output depending on the values of **RequestPassThru** and **SmartCardToMF** in the NFuse.conf file.

Support for ticketing is not available in the first release of the Citrix XML Service for MetaFrame for UNIX Operating Systems servers. Support for ticketing is available for servers running MetaFrame for UNIX Operating Systems, Feature Release 1.

NFuse_TicketUpper

Retrieves first 14 characters of an authentication ticket. Used to replace password information in an ICA file. This can be used to place a ticket in an ICA file that contains a static user name or user name supplied by another NFuse Classic tag.

Example

```
User=RCCollins
Domain=[NFuse_TicketLower]
ClearPassword=[NFuse_TicketUpper]
```

NFuse_TicketLower

Retrieves last 16 characters of an authentication ticket preceded by a backslash. Can be used to place a ticket in an ICA file that contains a static user name or user name supplied by another NFuse Classic tag.

Example

```
User=RCCollins
Domain=[NFuse_TicketLower]
ClearPassword=[NFuse_TicketUpper]
```

NFuse_TicketTimeToLive

Specifies the amount of time during which a ticket is valid. During this period the ticket can be used once. After the time period passes, the ticket is no longer valid. Values are specified in seconds.

NFuse_GroupNames

Retrieves a colon (:) delimited list of group names. When using group names to filter retrieved applications, the TemplateParser obtains the list of group names from this session field.

Example

```
Domain Users:Administrators:Public
```

NFuse_useCredentialType

Specifies the type of user credentials to use to retrieve applications. The default value is UseActualCredentials.The TemplateParser always checks this session field when passing credentials to the CitrixWireGateway object.

Values

UseActualCredentials
Use credentials composed of a user name, domain, and password.

UseGroupCredentials
Use a group ID. See “GroupCredentials” on page 44.

UseNullCredentials
Use a credential set composed of a null user name, null domain, and null password. Use this method to retrieve all published applications in a server farm.

NFuse_ContentType

Specifies the MIME content type the Web server reports for the response.

NFuse_MIMEExtension

Included to support older browsers that use file extensions for MIME type mapping. Use if your browser requires that your URLs end in .ica in order for the URL to be associated with an ICA Client. This tag does not function as a session field. The tag must always be equal to the value “.ica”.

Example

```
<a href=“launch.asp?NFuse_Application=Excel&NFuse_MIMEExtension=.ica”>
```

NFuse_TemplatesDir

Specifies the directory in which the template ICA or HTML file (as specified by the **NFuse_Template** session field) can be found. This path must be absolute to the machine on which the Web server is running, such as **C:\InetPub\wwwroot\NFuse**. Do not use Web server relative paths.

NFuse_TemplatesURL

This tag specifies the directory in which the template HTML or ICA file (as specified by the **NFuse_Template** session field) can be found. This path is relative to your Web server's Web root directory.

NFuse_SetSessionField

Sets a session field from within a template ICA or HTML file. When parsing an NFuse_SetSessionField tag, the TemplateParser object removes the tag itself after setting the specified session field.

Syntax

The NFuse_SetSessionField session field uses the following syntax:

```
[NFuse_SetSessionField SessionField=Value]
```

Values

SessionField
The name of the session field to set.

Value
The new value for the session field.

NFuse_Cache

Controls caching of application information on the Web server.

Values

NoCache
Do not cache App object data.

UseCache
Use cached App data first before checking the MetaFrame server for application information. If the data does not exist in the cache or the data is expired, NFuse Classic contacts the MetaFrame server for application information. This value is the default value.

RefreshCache
Contact the MetaFrame server for application information and update the App object data in the cache.

NFuse_ClientLogon

This tag will be replaced with the string **DisableCtrlAltDel=Off** or left blank, depending on the **SmartCardToMF** setting in the NFuse.conf file. The string prevents the bypass of the MetaFrame server's intial Ctrl-Alt-Del dialog (as required for Smart Cards).

NFuse_AppCommandLine

This tag will be replaced with command line paramters when launching content, such as the path of the file containting the content.

NFuse_LogonMode

This tag is designed to be set prior to calling the Template Parser and indicates the mode used when logging into NFuse.

Values

Explicit
Explicit credentials are presented.

Guest
Anonymous logon is used.

SmartCard
Smart Card logon.

Integrated
Integrated or Desktop Credential Pass-through logon.

NFuse_SOCKSSettings

This tag will be replaced by appropriate values of **ICASOCKSProtocolVersion**, **ICASOCKSProxyHost**, and **ICASOCKSProxyPortNumger** if the **SOCKSProxy** setting is present in the NFuse.conf file. If the **SOCKSProxy** setting is not present, this will be left blank.

Application Property Tags

Application property tags represent properties of a published application. The application whose properties replace these tags is determined by the **NFuse_Application**, **NFuse_CitrixServer**, **NFuse_User**, **NFuse_Domain**, and **NFuse_Password** session fields.

All of the following properties are initially set by a Citrix administrator at the time of application publishing. You can explicitly override these properties by using a set session field command, in which case the overriding value is used.

NFuse_WindowType

Specifies the window type of the ICA session window for the referenced application.

Values

percent
: Size of the ICA session window as a percentage of the client desktop.

pixels
: Size of the ICA session window in pixels.

seamless
: Specifies the application is to be run as a seamless window.

fullscreen
: Specifies the application is to be run as a full screen.

NFuse_WindowHeight

Retrieves the height, in pixels, of the ICA session window for the referenced application.

NFuse_WindowWidth

Retrieves the width, in pixels, of the ICA session window for the referenced application.

NFuse_WindowScale

Retrieves the percentage of the client device's desktop that the ICA session window should occupy.

Return Value

This tag returns an integer from 0 to 100. If no percentage size is specified for this application, the tag returns 0.

NFuse_WindowColors

Retrieves an integer representing the number of colors used in the ICA session window to display the referenced application.

Return Value

1
16 colors

2
256 colors

4
High colors

8
True colors

NFuse_IcaWindow

Retrieves ICA session window information for placement in ICA files. The TemplateParser object replaces this tag with all window information required in an ICA file.

NFuse_EncryptionLevel

Retrieves a string representing the level of encryption to use for the referenced application. To enable encryption levels higher than Basic, the MetaFrame server must support RC5 encryption. Support for RC5 encryption is included in MetaFrame XP, Feature Release 1 for MetaFrame 1.8, and SecureICA Services. MetaFrame for UNIX Operating Systems servers do not support RC5 encryption.

Return Values

basic
Basic encryption (XOR)

rc5-login
128-bit encryption for login only

rc5-40
40-bit encryption

rc5-56
56-bit encryption

rc5-128
128-bit encryption

NFuse_IcaEncryption

Retrieves encryption information for placement in ICA files. The TemplateParser object replaces this tag with all encryption information required in an ICA file.

NFuse_SoundType

Retrieves a string representing the level of audio support for the referenced application.

Return Values

none
No sound support

basic
Sound 1.0

NFuse_VideoType

Retrieves an integer representing the level of video support to use for the referenced application.

Return Values

none
No video support

basic
Video 1.0

NFuse_AppFriendlyName

Retrieves the friendly name (also called *Display Name* or *External Name*) of an application published in a MetaFrame server farm. Friendly names identify applications to client device users. Use friendly names to display application names to users; for example in an application list page. This property cannot be set as a session field.

NFuse_AppFriendlyNameUrlEncoded

Retrieves the friendly name (also called *Display Name* or *External Name*) of an application published in a MetaFrame server farm, returning the data in a URL-encoded string. This format is useful when the returned string must be placed in a URL, cookie, or other location where certain characters, if left unencoded, can cause errors in a Web browser or Web server.

NFuse_AppName

Retrieves the internal application name (also called *Application Name* or *Application ID*) of an application published in a MetaFrame server farm.

MetaFrame servers use these application names internally to identify applications. A single internal name cannot be used by more than one application. Use internal names when identifying an application to run; for example, in an ICA file initial program entry such as **InitialProgram=#[NFuse_AppName]**.

This property cannot be set as a session field.

NFuse_AppNameUrlEncoded

Retrieves the internal application name (also called *Application Name* or *Application ID*) of an application published in a MetaFrame server farm, returning the data in a URL-encoded string. This format is useful when the returned string must be placed in a URL, cookie, or other location where certain characters, if left unencoded, can cause errors in a Web browser or Web server.

MetaFrame servers use these application names internally to identify applications. A single internal name cannot be used by more than one application. Use internal names when identifying an application to run; for example, in an ICA file initial program entry such as **InitialProgram=#[NFuse_AppName]**.

This property cannot be set as a session field.

NFuse_IPv4Address

Retrieves the IP address of the MetaFrame server hosting the published application.

The use of this tag relates to the **NFuse_ClientName** tag described below. If you do not use **NFuse_ClientName** to specify a client name for the client connecting to the published application, **NFuse_IPv4Address** generates a unique client name based on the credentials contained in the **NFuse_User**, **NFuse_Password**, and **NFuse_Domain** session fields.

NFuse_IPv4AddressAlternate

Retrieves the external (or public) IP address of the MetaFrame server hosting the published application. Use this tag when accessing a MetaFrame server through a firewall.

The use of this tag relates to the **NFuse_ClientName** tag described below. If you do not use **NFuse_ClientName** to specify a client name for the client connecting to the published application, **NFuse_IPv4AddressAlternate** generates a unique client name based on the credentials contained in the **NFuse_User**, **NFuse_Password**, and **NFuse_Domain** session fields.

To use alternate addressing, you must also configure the MetaFrame server. If your server is a MetaFrame for Windows server, see your server documentation for information about using the **ALTADDR** utility. For MetaFrame for UNIX Operating Systems servers, see your server documentation for information about using the **ctxalt** utility.

NFuse_ClientName

Retrieves a unique client name for the ICA Client based on the credentials contained in the **NFuse_User**, **NFuse_Password**, and **NFuse_Domain** session fields. A client name is required for session reconnect and client device mapping.

Note NFuse Classic generates the client name for ICA connections from the username and domain of the user, and is visible in the Citrix Management Console. However, this may appear in the Console in a shortened form and may contain characters not present in the original username or domain. This ensures unique client names and does not indicate a problem.

NFuse_AppDescription

Retrieves the description for the referenced application.

NFuse_AppIcon

Retrieves the URL of the .Gif file for the referenced application. By default, when you use NFuse Classic to access applications, the NFuse Classic Java Objects create a .Gif file for each application. NFuse Classicsaves these files in a directory on your Web server.

NFuse_AppIconUrlEncoded

Retrieves the URL of the .Gif file for the referenced application. By default, when you use NFuse Classic to access applications, the Java Objects create a .Gif file for each application. NFuse Classic saves these files in a directory on your Web server.

This tag returns data in a URL-encoded string. This format is useful when the returned string must be placed in a URL, cookie, or other location where certain characters, if left unencoded, can cause errors in a Web browser or Web server.

User Interface Tags

The *user interface tags* create the application lists users see in their Web pages. User interface tags occur in pairs with the first tag opening the draw function and the second tag closing it. Text occurring between interface tags is parsed repeatedly for every application and folder in the user's view of the Program Neighborhood. After parsing of the interface tags completes, the TemplateParser object removes the interface tags themselves so that they are not seen by client Web browsers.

User interface tags are not settable as session fields.

NFuse_DrawPN

Opens the draw Program Neighborhood process. This tag takes a number of arguments that determine the look of what is drawn.

Values

NumCols=*n*
Draw application list in *n* columns.

NumRows=*n*
Draw application list in *n* rows.

Flat=[Yes | No]
Yes: Draw all the applications contained in the current folder and its subfolders. Draw no subfolders.

No: Draw only those applications and subfolders contained in the current folder.

Example

```
[NFuse_DrawPN NumCols="3" NumRows="3" flat="no"]
```

/NFuse_DrawPN

Closes the draw Program Neighborhood process.

Conditional Tags

Interface tags often appear with conditional tags. *Conditional tags* envelop text that is processed only if a particular condition is met. If the condition is met, the conditional tags themselves are removed and the TemplateParser object processes all text as normal. If the conditions are not met, the conditional tags themselves and all the text within them are removed and not processed. It is possible to nest one conditional tag within another.

Important All conditional tags, with the exception of the NFuse_IfSessionField and /NFuse_IfSessionField tags, must be used only during the draw Program Neighborhood process (between the NFuse_DrawPN and /NFuse_DrawPN tags). You can use the NFuse_IfSessionField and /NFuse_IfSessionField tags in any location in your pages.

Like the user interface tags, conditional tags are not settable as session fields.

NFuse_IfApp

Opening conditional tag. The text between this tag and the corresponding **/NFuse_IfApp** is processed and passed through only if the current item being enumerated between the **NFuse_DrawPN** and **/NFuse_DrawPN** tags is an App object, not a folder.

/NFuse_IfApp

Closing conditional tag.

NFuse_IfFolder

Opening conditional tag. The text between this and the corresponding **/NFuse_IfFolder** is processed and passed through only if the current item being enumerated between the **NFuse_DrawPN** and **/NFuse_DrawPN** tags is a folder, not an App object.

/NFuse_IfFolder

Closing conditional tag.

NFuse_IfRowStart

Opening conditional tag. The text between this tag and the corresponding **/NFuse_IfRowStart** is processed and passed through only if the current iteration of the DrawPN text is the first iteration in a row as determined by the numRows or numColumns arguments to the **NFuse_DrawPN** tag.

/NFuse_IfRowStart

Closing conditional tag.

NFuse_IfRowEnd

Opening conditional tag. The text between this tag and the corresponding **/NFuse_IfRowEnd** is processed and passed through only if the current iteration of the DrawPN text is the last iteration in a row as determined by the numRows or numColumns arguments to the **NFuse_DrawPN** tag.

/NFuse_IfRowEnd

Closing conditional tag.

NFuse_IfSessionField

Opening conditional tag. The text between this tag and the corresponding / NFuse_IfSessionField is processed and passed through only if the specified session field is currently set to the specified value.

The NFuse_IfSessionField conditional tag uses the following syntax:

```
<[NFuse_IfSessionField sessionfield=X value=Y]>
Code to process
<[/NFuse_IfSessionField]>
```

where *X* is the name of a session field, *Y* is the value you are testing for, and *Code to process* is some HTML, script, or substitution tags to include if the session field is currently set to *Y.*

/NFuse_IfSessionField

Closing conditional tag.

Citrix Secure Gateway

These are session fields and not intended for use as replacement tags. They are used to configure and direct the operation of the Citrix Secure Gateway (CSG).

NFuse_CSG_Server

This tag specifies the address of the Citrix Secure Gateway server. It must be a fully qualified domain name and not an IP address.

NFuse_CSG_ServerPort

This tag specifies the port corresponding to the server listed in **NFuse_CSG_Server**.

NFuse_CSG_Enable

This tag enables the use of the Citrix Secure Gateway. Any value other than **On** disables the use of the Citrix Secure Gateway.

NFuse_CSG_STA_URL

Use to define a comma-delimited list of servers providing Secure Ticketing Services. Each server should be specified as a URL.

NFuse_CSG_Address_Translation

This tag contorols which address of the MetaFrame server is used by the Citrix Secure Gateway.

Values

Normal
Uses the normal address of the MetaFrame server.

Alternate
Uses the alternate address of the MetaFrame server.

CHAPTER 3

Java Object Reference

The NFuse Classic objects are Java objects that you can access from Web server scripts or custom-written Java servlets to perform NFuse-related tasks. Use the Java objects perform to:

- Authenticate users to a MetaFrame server farm
- Retrieve per-user application sets from a server farm
- Modify the properties of individual applications before presenting them to users
- Parse template HTML and ICA files that display application sets to users and provide them with links to initiate ICA sessions

The Java objects that perform these tasks are:

- **CitrixWireGateway**. Creates a communication link between the Web page requesting a user's application information and the server farm containing that information.
- **ClearTextCredentials**. Encapsulates user, passworrd, and domain-style credentials. These may be used to represent a user account for NDS, NT, and UNIX systems.
- **AnonCredentials**.Used to represent anonymous (guest) users for authentication purposes.
- **GroupCredentials**. Contains a list of group names and an associated domain for use in retrieving applications for user groups.
- **AppEnumerator**. Provides an interface for accessing a user's application set.
- **App**. Represents a single application in an application set. App objects contain the properties of an application.
- **AppSettings**. Contains application properties you can modify.
- **AppDataList**. Contains a list of App objects that you can access quickly to determine application set lists for users.

- **AppListCache**. Caches AppDataList objects on a Web server so that you can retrieve application set information without repeatedly contacting the MetaFrame server farm.
- **TemplateParser**. Performs substitution tag processing on text files. This symbol substitution allows you to create template HTML and ICA files that TemplateParser modifies for each user.
- **AuthenticatedCredentials**. Encapsulates authentication information, such as the User Principle Name (UPN) or the Security Identifier (SID) for presentation to the server farm. This provides NFuse Classic with the ability to authenticate users using certificates. This functionality is only available on the Internet Information Services (IIS) 5 and 6 platforms, and requires use of Advanced Directory Services.
- **PropertiesKeys**. Provides scripts with an interface to the NFuse.conf file.

NFuse Classic Java Objects

The following sections include descriptions of the included Java objects. Each section concludes with code examples describing how to create the objects and call their methods.

CitrixWireGateway

A CitrixWireGateway object establishes a communication link between a Web server script and a MetaFrame server farm. You use a CitrixWireGateway object to create a communication channel through which you can send the server farm a user's credentials and receive in return application information for that user.

CitrixWireGateway objects have the following methods, which initialize the CitrixWireGateway. These methods must be called prior to calling **getAppEnumerator()** on the CitrixWireGateway object:

void initialize(Credentials *credentials*)

This version of the method directs the initialization request to the default MetaFrame server, on the default TCP/IP port, over the default protocol specified in the NFuse Classic configuration file.

If the default protocol is not specified in the configuration file or if it is set to "HTTP," the method creates an HTTP connection to the default server on the default port. If the protocol specified in the configuration file is "SSL," the method determines the SSL Relay server and port specified in the properties file and creates an SSL connection through the SSL Relay server. By default, the configuration file does not contain entries for SSL Relay server, relay port, or protocol. You can add these values to the configuration file.

void initialize(Credentials *credentials*, String *citrixServer*, String *transport*, int *citrixServerPort*)

Use this version of the Initialize() method to direct an initialization request to a MetaFrame server or TCP/IP port that differs from the default server and port specified in the configuration file.

void initialize(Credentials *credentials*, String *citrixServer*, String *transport*, int *citrixServerPort*, String *relayServer*, int *relayServerPort*)

Use this version of the Initialize() method to direct an initialization request to a non-default MetaFrame server through a Citrix SSL Relay server or SSL port that differs from the default SSL Relay server and port specified in the properties file. By default, the configuration file does not contain entries for SSL Relay server, relay port, or protocol. You can add these values to the configuration file.

void initialize(Credentials *credentials*, String *relayServer*, int *relayServerPort*, String *transport*, int *reserved*)

Use this version of the Initialize() method if you want to direct your initialization request to a default MetaFrame server through a Citrix SSL Relay server or SSL port that differs from the default SSL Relay server and port specified in the properties file. By default, the configuration file does not contain entries for SSL Relay server, relay port, or protocol. You can add these values to the configuration file.

Note Support for the Citrix SSL Relay is not available in the first release of the Citrix XML Service for MetaFrame for UNIX Operating Systems servers. MetaFrame for UNIX, Feature Release 1, does support the Citrix SSL Relay.

Parameters

credentials

A Credentials object. A CitrixWireGateway object uses these credentials to authenticate the user with the MetaFrame server maintaining the application information and to filter the applications returned by the getAppEnumerator() method.

citrixServer

The DNS name, NetBIOS name, or IP address of the MetaFrame server from which to retrieve the application information.

transport
The protocol over which to transport the NFuse Classic data. NFuse Classic supports two protocols: HTTP (Hypertext Transport Protocol) and SSL (Secure Socket Layer). Use HTTP to send the NFuse Classic data over a standard HTTP connection. Specify SSL to send data over a secure connection that uses a MetaFrame server running the Citrix SSL Relay to perform host authentication and data encryption. Specify HTTPS to send data over a secure connection via SSL.

citrixServerPort
The TCP/IP port on which citrixServer listens for NFuse Classic requests.

relayServer
The DNS name, NetBIOS name, or IP address of a MetaFrame server running the Citrix SSL Relay.

relayServerPort
The port on which relayServer listens for SSL requests.

reserved
Always set to 0.

Return

None.

AppEnumerator getAppEnumerator()

Retrieves an AppEnumerator object for a CitrixWireGateway object. You can then use the returned AppEnumerator object to enumerate the App objects retrieved by the CitrixWireGateway object.

Before calling this method, you must call the initialize() method on the CitrixWireGateway object.

Parameters

None.

Return

An AppEnumerator object or null if there was an error in communication with the MetaFrame server.

App getApp(String *desiredAppName*)

Retrieves a single specific App object from a CitrixWireGateway object. Use this method instead of getAppEnumerator() if you want to retrieve a single application instead of all applications for a specific user.

Before calling this method, you must call the initialize() method on the CitrixWireGateway object.

Parameters

desiredAppName

The name of the published application you want to retrieve.

Return

An App object or null if there was an error in communication with the MetaFrame server.

AppDataList getAppDataList()

Retrieves an AppDataList object from a CitrixWireGateway object. You can save the AppDataList object in an AppListCache object and reference it in other Web server sessions.

Parameters

None.

Return

An AppDataList object or null if there was an error in communication with the MetaFrame server.

String getLastError()

If the user of the CitrixWireGateway object detects that an error has occurred, getLastError() should be called to return a description of the error.

Parameters

None.

Return

A description of the last error on this object.

void setClientAddress(String *clientAddress*)

Sets the user's client address.

Parameter

clientAddress

The client address to be set.

Return

None.

void setClientAddressType(String *clientAddress*)

Sets the user's client address type.

Parameter

clientAddress

The client address type to be set. Valid values include **DNS, DNS-PORT, ipv4, ipv4-port, and URI.**

Return

None.

boolean changePassword(String *oldPassword*, String *newPassword*, String *confirmPassword*)

Use this method to change the password for the current user.

Parameters

oldPassword

The existing password for the current user.

newPassword

The new password to set for the current user.

confirmPassword

The new password again, to confirm it was entered correctly.

Return

Returns **True** if the password was changed, **False** if it was not changed. Use GetLastError to determine the error.

Example Usage: CitrixWireGateway Object

CitrixWireGateway object creation is a two-step process: first create a CitrixWireGateway object and then call the gateway's initialize() method.

The following Active Server Page/VBScript example illustrates how to create and initialize a CitrixWireGateway object. In this example, myCredentials is a previously created Credentials object.

```
    'create a CitrixWireGateway object
set myGat = Server.CreateObject("com.citrix.nfuse.CitrixWireGateway")
    'initialize the gateway object
myGat.initialize myCredentials
```

The above example initializes the gateway with a Credentials object and unspecified citrixServer, transport, and citrixServerPort parameters, thus causing the Java objects to look in the NFuse Classic configuration file for defaults for the unspecified parameters. The following example illustrates

how to override the defaults in the configuration file. Once again, myCredentials is a previously created Credentials object.

```
    ‘initialize the gateway object so that the Web server
    ‘contacts a MetaFrame server named ServerX on TCP port 9999
myGat.initialize myCredentials, ServerX, "Http", 9999
```

The next example illustrates how to create a secure connection through a MetaFrame server running the Citrix SSL Relay. Once again, myCredentials is a previously created Credentials object.

```
    ‘initialize the gateway object so that the Web server
    ‘contacts a MetaFrame server named ServerX on TCP port 9999 through an SSL Relay server named
    ‘ServerR on port 443
myGat.initialize myCredentials, ServerX, "SSL", 9999, ServerR, 443
```

The above example explicitly specifies the name of the MetaFrame server running the Citrix SSL Relay and the port on which it is listening for requests. If these parameters are not specified, the Initialize() method searches the configuration file for default values.

ClearTextCredentials

A ClearTextCredentials object is a container that holds an ICA Client user's credentials. User credentials include the user's user name and Windows NT domain in plain text and a password encrypted using basic encryption. Use a ClearTextCredentials object to package a user's credentials before sending those credentials within a CitrixWireGateway object to the server farm for authentication.

ClearTextCredentials objects have the following methods:

void initialize(String *user*, String *domain*, String *password*)

This version of the method uses user name, domain, and password information to initialize a ClearTextCredentials object. You must call this method before you use the ClearTextCredentials object or pass it to another object (such as CitrixWireGateway). When authenticating to MetaFrame for UNIX Operating Systems servers, can include a domain name to use during initialization, or specify an empty domain by adding the line **ForceLoginDomain=** to the NFuse.conf file.

void initialize()

Use this version of the Initialize() method if to initialize a ClearTextCredentials object without user name, domain, and password information (also called a null credential authentication request). This form of initialization lets you retrieve all published applications for all users in the server farm.

Parameters

user
The plain-text user name.

domain
The plain-text domain.

password
The plain-text password.

Return

None.

String getLastError()

If the user of the ClearTextCredentials object detects that an error has occurred, getLastError() should be called to return a description of the error.

Parameters

None.

Return

A description of the last error on this object.

void setDomainType (String *domainType)*

Sets the domain type for the ClearTextCredentials object. Use to generate an application list or launch an application. The domain type is sent to the Citrix XML relay service to distinguish between Windows NT, Novell Directory Services and UNIX logins.

Parameters

domainType

Valid domain types include **NT**, **NDS**, and **UNIX**. If null or an empty string, the default domain type will be set to **NT**

Return

None.

String getDomainType *()*

Returns the domain type of the ClearTextCredentials object it is called on.

Parameters

None.

Return

Returns the domain type set by setDomainType or **NT** if setDomainType has not been called.

Returns null for uninitialized ClearTextCredentials objects.

Example Usage: ClearTextCredentials Object

ClearTextCredentials object creation is a two-step process: first create a ClearTextCredentials object, then call the credential's object initialize() method.

The following Active Server Page/VBScript example illustrates how to create and initialize a ClearTextCredentials object. In this example, user, domain, and password are variables representing previously collected user authentication information.

```
    'create a ClearTextCredentials object
Set myC = Server.CreateObject("com.citrix.nfuse.Credentials")
    'initialize the credentials object
myC.initialize user, domain, password
```

GroupCredentials

A GroupCredentials object is a container that holds a list of group names and an associated domain. Use a GroupCredentials object to retrieve a list of applications for a user group using a CitrixWireGateway object.

Note GroupCredentials object provides a viewable application list only. You cannot use a GroupCredentials object to authenticate users or groups to a farm.

GroupCredentials objects have the following methods:

void setDomain(String *domain*)

Populates a GroupCredential object's domain field.

Parameters

domain

The domain name associated with the set of group names.

Return

None.

void addGroupName(String *newGroupName*)

Adds the specified group name to the list of group names associated with the groupCredentials object.

Parameters

newGroupName

The group name to add to the group names list.

Return

None.

void setGroupNames(String[ ] *newGroupNamesArray*)

Populates the GroupCredentials object with a new set of group names contained in an array. This method replaces all previously added group names (if any).

Parameters

newGroupNamesArray

Array containing a list of names to use to populate the GroupCredentials object.

Return

None.

void setGroupNames(Vector *newGroupNamesVector*)

Like setGroupNames(), this method populates the GroupCredentials object but uses vectors instead of an array of strings.

Parameters

newGroupNamesVector

Vector containing a list of names to use to populate the GroupCredentials object.

Return

None.

String getFirstGroupName()

Returns the first group name associated with the GroupCredentials object. Use to obtain a list of all groupnames associated with the GroupCredentials object. Subsequent call can be getNextGroupName().

Parameters

None.

Return

First group name associated with the GroupCredentials object.

String getNextGroupName()

Returns the next available group name associated with the GroupCredentials object. Called after getFirstGroupName(). Returns null if no more group names are found.

Parameters

None.

Return

Next group name in the group name list.

int getGroupNamesListSize()

Returns the number of group names associated with the GroupCredentials object.

Parameters

None

Return

The total number of group names associated with the GroupCredentials object.

String getGroupNameAt(int *index*)

Returns a group name at a particular index in the list of group names associated with the GroupCredentials object.

Parameters

index

Position in the group name list.

Return

Group name at the specified index.

void setDomainType(String *domainType*)

Sets the domain type for the GroupCredentials object. Possible types include NT and UNIX.

Parameters

domainType

Type of domain used for group name association.

Return

None.

String getDomainType()

Returns the current domain type associated with the GroupCredentials object. Possible types include NT and UNIX.

Parameters

None.

Return

Type of domain currently associated with the GroupCredentials object.

String getDomain()

Returns the domain currently associated with the GroupCredentials object.

Parameters

None.

Return

Domain currently associated with the GroupCredentials object.

String getLastError()

If the user of the GroupCredentials object detects that an error has occurred, getLastError() should be called to return a description of the error.

Parameters

None.

Return

A description of the last error on this object.

Example Usage: GroupCredentials Object

The following Active Server Page/VBScript example illustrates how to create a GroupCredentials object. The example then uses the addGroupName() method to add a group named myGroup to the object.

```
    ‘create a GroupCredentials object
Set myCredentials = Server.CreateObject("com.citrix.nfuse.GroupCredentials")
    ‘add a group name to the GroupCredentials object
myCredentials.addGroupName("myGroup")
```

AppEnumerator

An AppEnumerator object is a class that your Web pages can use to access applications published in a server farm. AppEnumerator returns App objects and Program Neighborhood folders that you can then use to manipulate applications. AppEnumerator is returned by a CitrixWireGateway object (see “CitrixWireGateway” on page 36).

Citrix Management Console and Published Application Manager (the tools used by Citrix administrators to publish applications on MetaFrame XP and 1.8 servers, respectively) organizes applications in a folder hierarchy to give administrators the ability to present users with applications grouped in folders. When enumerating applications or subfolders for inclusion in your Web pages, you must specify a base folder from which to start the enumeration. Applications can either be enumerated flatly or normally. A *flat* enumeration causes all the applications contained in the base folder and any of its subfolders to be returned in the enumeration. A *normal* enumeration causes only those applications directly contained in the base folder to be returned in the enumeration.

AppEnumerator also has methods for enumerating folders. All folder enumeration methods are normal.

AppEnumerator objects have the following methods:

App nextAppFlat(String *folder*)

Returns the next application in the current flat enumeration. When using this method to cycle through applications, all applications contained in the specified folder and its subfolders are returned.

Note A new flat app enumeration is started when a folder name is passed to this method or hasMoreAppsFlat() that is different than the previous folder name passed to this method or hasMoreAppsFlat().

Parameters

folder

The folder in which to enumerate applications.

Return

An App object for an application contained in the specified folder (or any of its subfolders) or null if there are no more applications to be enumerated in the specified folder or its subfolders.

boolean hasMoreAppsFlat(String *folder*)

Determines if there are more applications to enumerate in the current flat app enumeration. That is, this method reports whether or not there are more applications to enumerate in the specified folder or any of its subfolders.

A new flat enumeration is started when a folder name is passed to this method or nextAppFlat() that is different than the previous folder name passed to this method or nextAppFlat().

Parameters

folder

The folder in which to determine if there are more applications to enumerate.

Return

True: There are more application(s) in the specified folder (or any of its subfolders) to enumerate.

False: There are no more applications in the specified folder (or any of its subfolders) to enumerate.

int getNumAppsFlat(String *folder*)

Returns the total number of applications contained in the specified folder and its subfolders.

Does not affect the current flat app enumeration.

Parameters

folder

The folder in which to determine the number of applications.

Return

The total number of applications contained in the specified folder and its subfolders. This number does not include folders contained in the specified folder; for example, if the specified folder contains one application and a subfolder that contains an application, this method returns 2 (the subfolder is not counted as an application).

App nextApp(String *folder*)

Returns the next application in the current normal enumeration. When using this method to cycle through applications, only those applications directly contained in the specified folder are returned.

A new normal enumeration is started when a folder name is passed to this method or hasMoreApps() that is different than the previous folder name passed to this method or hasMoreApps().

Parameters

folder

The folder in which to enumerate applications.

Return

An App object for an application contained in the specified folder or null if there are no more applications to enumerate in the specified folder.

boolean hasMoreApps(String *folder*)

Determines if there are more applications to enumerate in the current normal enumeration; that is, this method reports whether or not there are more applications to enumerate in the specified folder (but not any of its subfolders).

A new normal enumeration is started when a folder name is passed to this method or nextApp() that is different than the previous folder name passed to this method or nextApp().

Parameters

folder

The folder in which to determine if there are more applications to enumerate.

Return

True: There are more application(s) to enumerate in the specified folder.

False: There are no more applications to enumerate in the specified folder.

int getNumApps(String *folder*)

Returns the number of applications immediately contained in the specified folder.

Does not affect the current normal app enumeration.

Parameters

folder

The folder in which to determine the number of applications.

Return

The number of applications immediately contained in the specified folder. This number includes neither the folders in the specified folder nor applications contained in subfolders of the specified folder; for example, if the specified folder contains one application and a subfolder that contains an application, this method returns 1.

String nextFolder(String *folder*)

Returns the next folder in the current folder enumeration. Only those folders directly contained in the specified folder are returned.

A new folder enumeration is started when a folder name is passed to this method or hasMoreFolders() that is different than the previous folder name passed to this method or hasMoreFolders().

Parameters

folder

The folder in which to enumerate subfolders.

Return

A folder immediately contained in the specified folder in the form "\aaa\bbb\ccc\…" or null if there are no more folders to enumerate in the specified folder.

String nextFolderUrlEncoded(String *folder*)

Returns the next folder in the current folder enumeration. Returned data is in a URL-encoded string. This format is useful when the returned string must be placed in a URL, cookie, or other location where certain characters, if left unencoded, can cause errors in a Web browser or Web server.

Only those folders directly contained in the specified folder are returned.

A new folder enumeration is started when a folder name is passed to this method or hasMoreFolders() that is different than the previous folder name passed to this method or hasMoreFolders().

Parameters

folder

The folder in which to enumerate subfolders.

Return

A folder immediately contained in the specified folder in the form "%5Caaa%5Cbbb%5Cccc%5C…" or null if there are no more folders to enumerate in the specified folder.

boolean hasMoreFolders(String *folder*)

Determines if there are more folders to enumerate in the current folder enumeration; that is, this method reports whether or not there are more subfolders immediately contained in the specified folder.

A new folder enumeration is started when a folder name is passed to this method or nextFolder() that is different than the previous folder name passed to this method or nextFolder().

Parameters

folder

The folder in which to enumerate folders.

Return

True: There are more folder(s) to enumerate in the specified folder.

False: There are no more folders to enumerate in the specified folder.

int getNumFolders(String *folder*)

Returns the number of folders immediately contained in the specified folder.

Does not affect the current folder enumeration.

Parameters

folder

The folder in which to determine the number of subfolders.

Return

The number of folders immediately contained in the specified folder. This number does not include applications in the specified folder; for example, if the specified folder contains one application and a subfolder, this method returns 1.

String getLastError()

If the user of the AppEnumerator object detects that an error has occurred, getLastError() can be called to return a description of the error.

Parameters

None.

Return

A description of the last error on this object.

Example Usage: AppEnumerator Object

The following Active Server Page/VBScript example illustrates how to create an AppEnumerator object and then how to retrieve a single App object from AppEnumerator. In this example, myGat is a previously created CitrixWireGateway object.

```
    ‘create an AppEnumerator object
Set myEnumerator = myGat.getAppEnumerator
    ‘retrieve the first App object from the AppEnumerator object
Set myApp = myEnumerator.nextAppFlat("")
```

App

An App object is a container that holds the properties of a single application published in a MetaFrame server farm. You use an App object to access the results of a query of a server farm for information such as a published application's name, the size and color depth of its ICA session window, and its supported encryption level, among other properties set by a MetaFrame server administrator at the time of application publishing.

An App object gives you query access to both an application's non-settable and settable properties. *Non-settable properties* include properties that define an application (such as the application name) and properties that can be determined only by the publisher of the application (such as if the application is published only on TCP/IP machines or if the application is disabled).

Settable properties are application properties that Web masters can modify in their Web server scripts (see "AppSettings" on page 74 for information about modifying settable properties).

You create App objects by retrieving them from an AppEnumerator object (see "AppEnumerator" on page 47).

App objects have the following methods:

String getFriendlyName()

Retrieves the friendly name (also called Display Name or External Name) of an application published in a MetaFrame server farm. Friendly names identify applications to Program Neighborhood users.

Use friendly names to display application names to users; for example in an application list page.

This property is non-settable.

Parameters

None.

Return

The friendly name of the application.

String getFriendlyNameUrlEncoded()

Retrieves the friendly name (also called Display Name or External Name) of an application published in a MetaFrame server farm. Return data is in a URL-encoded string. This format is useful when the returned string must be placed in a URL, cookie, or other location where certain characters, if left unencoded, can cause errors in a Web browser or Web server.

This property is non-settable.

Parameters

None.

Return

The friendly name of the application in URL-encoded format.

String getName()

Retrieves the internal application name (also called Application Name or Application ID) of an application published in a MetaFrame server farm. MetaFrame servers use these application names internally to identify applications. A single internal name cannot be used by more than one application.

Use internal names when identifying an application to run; for example, in an ICA file initial program entry such as InitialProgram=#[NFuse_AppName].

This property is non-settable.

Parameters

None.

Return

The internal name of the application.

String getNameUrlEncoded()

Retrieves the internal application name (also called Application Name or Application ID) of an application published in a MetaFrame server farm. Return data is in a URL-encoded string. This format is useful when the returned string must be placed in a URL, cookie, or other location where certain characters, if left unencoded, can cause errors in a Web browser or Web server.

This property is non-settable.

Parameters

None.

Return

The internal name in URL-encoded format.

String getWindowType()

Returns a string representing the type of window in which the referenced application is set to display.

Parameters

None.

Return

pixels: Size specified as height and width in pixels

percent: Size specified as percentage of client desktop

seamless: ICA Client window is seamless

fullscreen: ICA Client window is full-screen mode

int getWindowPercentage()

Retrieves the percentage of the client device's desktop that the ICA session window should occupy. You can set this property using an AppSettings object.

Parameters

None.

Return

An integer from 0 to 100. If no percentage size is specified for this application, the method returns 0.

int getWindowWidth()

Retrieves the width in pixels of the ICA session window for the referenced application. You can set this property using an AppSettings object.

Parameters

None.

Return

A positive integer representing the width in pixels. If no height and width size are specified for this application, the method returns 0.

int getWindowHeight()

Retrieves the height in pixels of the ICA session window for the referenced application. You can set this property using an AppSettings object.

Parameters

None.

Return

A positive integer representing the height in pixels. If no height and width size are specified for this application, the method returns 0.

int getColorDepth()

Retrieves an integer representing the number of colors used in the ICA session window to display the referenced application. You can set this property using an AppSettings object.

Parameters

None.

Return

1: 16 colors

2: 256 colors

4: High colors

8: True colors

String getSoundType()

Retrieves a string representing the level of audio support for the referenced application. You can set this property using an AppSettings object.

Parameters

None.

Return

none: No sound support

basic: Sound 1.0

String getEncryptionLevel()

Retrieves a string representing the level of encryption to use for the referenced application. You can set this property using an AppSettings object. To enable encryption levels higher than Basic, the MetaFrame server must support RC5 encryption. Support for RC5 encryption is included in MetaFrame XP, Feature Release 1 for MetaFrame 1.8, and SecureICA Services. MetaFrame for UNIX Operating Systems servers do not support RC5 encryption.

Parameters

None.

Return

basic: Basic encryption (XOR)

rc5-login: 128-bit for login only

rc5-40: 40-bit

rc5-56: 56-bit

rc5-128: 128-bit

String getVideoType()

Retrieves a string representing the level of video support to use for the referenced application. You can set this property using an AppSettings object. Use of Video 1.0 requires Citrix VideoFrame.

Parameters

None.

Return

none: No Video support

basic: Video 1.0

boolean getStartMenu()

Retrieves a boolean representing whether the referenced application has the property set for placing a shortcut in the Windows Start menu of ICA Win32 Client devices. You can set this property using an AppSettings object.

Parameters

None.

Return

True: This application has the Start menu shortcut creation property set to on.

False: This application has the Start menu shortcut creation property set to off.

boolean getDesktop()

Retrieves a boolean representing whether the referenced application has the property set for placing a shortcut on the Windows desktops of ICA Win32 Client devices. You can set this property using an AppSettings object.

Parameters

None.

Return

True: This application has the desktop shortcut creation property set to on.

False: This application has the desktop shortcut creation property set to off.

String getFolder()

Retrieves the Program Neighborhood folder to which the referenced application belongs. You can set this property using an AppSettings object.

Parameters

None.

Return

The folder in the form "\aaa\bbb\…"

String getDescription()

Retrieves the description for the referenced application. You can set this property using an AppSettings object.

Parameters

None.

Return

The application's description.

String getIconFile()

Retrieves the URL of the .Gif file for the referenced application. By default, when you use NFuseClassic to access applications, the Java objects create a .Gif file for each application. NFuse Classic saves these files in a directory on your Web server. You can set this property using an AppSettings object.

Parameters

None.

Return

The location of the .Gif file relative to the Web server's root directory. That is, this method returns what can be placed to the right of SRC in the HTML tag <img SRC="*path*">.

String getIconFileUrlEncoded()

Retrieves the URL of the .Gif file for the referenced application. Return data is in a URL-encoded string. This format is useful when the returned string must be placed in a URL, cookie, or other location where certain characters, if left unencoded, can cause errors in a Web browser or Web server.

Parameters

None.

Return

A URL-encoded string representing the location of the .Gif file relative to the Web server's root directory. That is, this method returns what can be placed to the right of SRC in the HTML tag <img SRC="*path*">.

void applySettings(AppSettings *newSettings*)

Overrides a current property or properties of an application with new properties set using an AppSettings object. You must use this method to initialize any property changes for an application.

Note You can use this method to override application properties set at the time of application publishing or properties set by a previous instance of the applySettings method.

See "Example Usage: AppSettings Object" on page 79 for an example illustrating how to use the applySettings() method.

Parameters

newSettings

The new settings to be applied over the application's existing settings.

Return

None.

String getIPv4Address(CitrixWireGateway *gateway*)

String getIPv4Address(CitrixWireGateway *gateway*, String *clientname*)

Using a passed-in CitrixWireGateway object, this method retrieves the IP address of the MetaFrame server hosting the published application. If a client name is not specified, the method generates a unique client name based on the credentials contained in the gateway object. The client name is used to identify the client device connecting to the application.

By using this method instead of getName() to place an address in an ICA file, you can eliminate ICA Client-side UDP browsing used in name resolution of published applications.

Parameters

gateway
A CitrixWireGateway object.

clientname
A unique name to identify the ICA Client connecting to the application. Leave unspecified to cause the getIPv4Address() method to generate a client name.

Return

The IP address of the MetaFrame server hosting the application.

String getIPv4AddressAlternate(CitrixWireGateway *gateway*)

String getIPv4AddressAlternate(CitrixWireGateway *gateway*, String *clientname*)

Use this method to access a MetaFrame server across a firewall. Using a passed-in CitrixWireGateway object, this method retrieves the external (or public) IP address of the MetaFrame server hosting the published application. If a client name is not specified, the method generates a unique client name based on the credentials contained in the gateway object. The client name is used to identify the ICA Client connecting to the application.

By using this method instead of getName() to place an address in an ICA file, you can eliminate ICA Client-side UDP browsing for published application name resolution.

See your MetaFrame server documentation for information about server-side configuration of ICA connections using alternate addresses.

To use alternate addressing, you must also configure the MetaFrame server. If your server is a MetaFrame for Windows server, see your server documentation for information about using the ALTADDR utility. For MetaFrame for UNIX Operating Systems servers, see your server documentation for information about using the ctxalt utility.

Parameters

gateway
A CitrixWireGateway object.

clientname
A unique name to identify the ICA Client connecting to the application. Leave unspecified to cause the getIPv4AddressAlternate() method to generate a client name.

Return

The external IP address of the MetaFrame server hosting the application.

String generateClientName(Credentials *credentials*)

Use this method to generate a unique client name that you can use to identify the ICA Client connecting to the application. This method uses the same client-name-generating algorithm as getIPv4Address(*gateway*) and getIPv4AddressAlternate(*gateway*).

Parameters

credentials
A Credentials object.

Return

A unique name to identify the ICA Client connecting to the application.

String getTicket(CitrixWireGateway wireGateway, Credentials credentials, String ticketType)

This method retrieves an authentication ticket for an application. Before calling this method, you must call either the getIPv4Address() or getIPv4AddressAlternate() method on the App object.

Support for ticketing is available for servers running MetaFrame for UNIX Operating Systems, Feature Release 1.

Parameters

wireGateway
The CitrixWireGateway object from which to retrieve the ticket.

credentials
A Credentials object. A CitrixWireGateway object uses these credentials to authenticate the user during ticket retrieval.

ticketType
Identifies the type of ticket to retrieve. Must be “CtxLogon.”

Return

A 30 character ticket string.

String getTicketUpper(CitrixWireGateway wireGateway, Credentials credentials, String ticketType)

Retrieves first 14 characters of an authentication ticket. The first 14 characters of a ticket correspond to information you can place in an ICA file as the value for a ClearPassword parameter. See the description of [NFuse_TicketUpper] in “General Tags” on page 18.

Before calling this method, you must call either the getIPv4Address() or getIPv4AddressAlternate() method on the App object.

Parameters

wireGateway
The CitrixWireGateway object from which to retrieve the ticket.

credentials
A Credentials object. A CitrixWireGateway object uses these credentials to authenticate the user during ticket retrieval.

ticketType
Identifies the type of ticket to retrieve. Must be "CtxLogon."

Return

A string composed of the first 14 characters of the ticket; for example, 7456C5E8F56EBE.

String getTicketLower(CitrixWireGateway wireGateway, Credentials credentials, String ticketType)

Retrieves the last 16 characters of an authentication ticket preceded by a backslash. The last 16 characters of a ticket correspond to information you can place in an ICA file as the value for a domain parameter. See the description of [NFuse_TicketLower] in "General Tags" on page 18.

Before calling this method, you must call either the getIPv4Address() or getIPv4AddressAlternate() method on the App object.

Parameters

wireGateway
The CitrixWireGateway object from which to retrieve ticket.

credentials
A Credentials object. A CitrixWireGateway object uses these credentials to authenticate the user during ticket retrieval.

ticketType
Identifies the type of ticket to retrieve. Must be "CtxLogon."

Return

A string composed of a backslash (\) followed by the last 16 characters of the ticket; for example, \9AC643FBAA919ADC.

void setTicketTimeToLive(CitrixWireGateway wireGateway, int ticketTimeToLive)

This method sets the duration for which the ticket is valid. When this time period passes or the ticket is used, the ticket is invalid. Specify values in seconds.

Parameters

wireGateway
The CitrixWireGateway object to use to set the timeout period. Must be the same object used to retrieve the ticket.

ticketTimeToLive
Amount of time for which the ticket is valid.

Return

None.

boolean equals(Object *app*)

Determines whether the passed-in object is equal to this App object. Equivalence means the two App objects represent the same published application.

Parameters

app
A Java object.

Return

True: The passed-in object is equivalent to this App object.

False: The passed-in object is not equivalent to this App object.

String getLastError

If the user of an App object detects that an error has occurred, getLastError() can be called to return a description of the error.

Parameters

None

Return

A description of the last error on this object.

boolean allowCustomizeWindowSize()

This checks if the end user can customize window size of the published applications.

Parameters

None.

Return

Returns **True** if the user can customize the window size.

boolean allowCustomizeWindowColor()

Tests if the end user can customize window color depth of the published applications.

Parameters

None.

Return

Returns **True** if the user can customize the window color depth.

boolean allowCustomizeAudio()

Tests if the end user can customize the audio options of the published applications.

Parameters

None.

Return

Returns **True** if the user can customize the audio options.

boolean allowCustomizeEncryption()

Tests if the end user can customize the encryption level of the published applications.

Parameters

None.

Return

Returns **True** if the users can customize the encryption level.

boolean allowCustomizeSettings()

Tests if the end user can customize the settings of the published applications.

Parameters

None.

Return

Returns **True** if the user can customize the settings.

boolean allowGuestLogin()

Tests if the NFuse Classic Web site will allow users to login and launch published applications as guests, in addition to allowing registered users to login and launch published applications. This is superceded by **allowLoginType(loginType)** and **onlyLoginType(loginType)**.

Parameters

None.

Return

Returns **True** if the NFuse Classic Web site allows guest users to login and launch published applications.

boolean guestLoginOnly()

Tests if the NFuse Classic Web site will allow only users to login and launch published applications as guest users. This is superceded by **allowLoginType(loginType)** and **onlyLoginType(loginType)**.

Parameters

None.

Return

Returns **True** if the NFuse Classic Web site only allows users to login as guests.

String getForceLoginDomain()

Returns login domain specified in the configuration file as ForceLoginDomain property.

Parameters

None.

Return

Returns an empty string if such property is not specified.

String getAddressResolutionType()

Returns the address type the NFuse Classic Web site will request Citrix XML Service to send.

Parameters

None.

Return

Possible return values are IPv4, IPv4-port, dns, dns-port.

public boolean getAppIsContent()

Determines if an object is a published application or published content.

Parameters

None.

Return

True: the object is published content.

False: the object is a published application.

public String getContentAdr()

Returns the address of an object that is published content and not a published application.

Parameters

None.

Return

The address of the published content; for example, “http://www.mypage.com” or “file://my-machine/NFuse.txt”.

public String getcontentAdrUrlEncoded(boolean *isIE*)

Returns the UNC address of an object that is published content and not a published application. This method converts a UNC name to be friendly to Netscape browsers.

Parameters

isIE

If **False**, will convert the UNC name.

If **True**, will not convert the UNC name.

Return

The address of the published content.

public boolean getDisabled()

Determines if a published application or published content object has been enabled or disabled using the Citrix Management Console.

Parameters

None.

Return

Returns **True** if the published application or published content object is disabled. Returns **False** if the published application or published content object is enabled.

public string getPublisherName()

Parameters

None.

Return

Returns a string that is the name of the MetaFrame farm an App object belongs to.

public boolean getSSLEnabled()

Parameters

None.

Return

Returns **True** if the published application is SSL enabled. Returns **False** if the published application is not SSL enabled. SSL can be enabled and disabled for published applications in the Citrix Management Console.

public String getIPv4PortAddress(CitrixwireGateway *wireGateway*)

Parameters

wireGateway

The CitrixwireGateway object from which to retrieve the server address.

Return

Returns a string value that represents the ipv4:port form of the address of a load-balanced MetaFrame server that will host a published application represented by an App object. Generates a *clientName* using the credentials supplied to initialize the CitrixWireGateway object, and is used for load balancing and reconnecting to disconnected sessions. For more information about initializing the CitrixWireGateway, see “CitrixWireGateway” on page 36.

public String getIPv4PortAddressAlternate(CitrixwireGateway *wireGateway*)

wireGateway

The CitrixwireGateway object from which to retrieve the server address.

Return

Returns a string value that represents the alternate ipv4:port form of the address of a load-balanced MetaFrame server that will host a published application represented by an App object. Generates a *clientName* using the credentials supplied to initialize the CitrixWireGateway object, and is used for load balancing and reconnecting to disconnected sessions. For more information about initializing the CitrixWireGateway, see “CitrixWireGateway” on page 36.

public String getDNSAddress(CitrixwireGateway *wireGateway*)

Parameters

wireGateway

The CitrixwireGateway object from which to retrieve the server address.

Return

Returns a string value that represents the DNS name of a load-balanced MetaFrame server that will host a published application represented by an App object. Generates a *clientName* using the credentials supplied to

initialize the CitrixWireGateway object, and is used for load balancing and reconnecting to disconnected sessions. For more information about initializing the CitrixWireGateway, see "CitrixWireGateway" on page 36.

public String getDNSPortAddress(CitrixwireGateway *wireGateway*)

Parameters

wireGateway

The CitrixwireGateway object from which to retrieve the server address.

Return

Returns a string value that represents the DNS:port form name of a load-balanced MetaFrame server that will host a published application represented by an App object. Generates a *clientName* using the credentials supplied to initialize the CitrixWireGateway object, and is used for load balancing and reconnecting to disconnected sessions. For more information about initializing the CitrixWireGateway, see "CitrixWireGateway" on page 36.

public String getIPv4PortAddress(CitrixwireGateway *wireGateway*, String *clientName*)

Parameters

wireGateway

The CitrixwireGateway object from which to retrieve the server address.

clientName

The client name for load balancing and reconnecting disconnected sessions.

Return

Returns a string value that represents the ipv4:port form of the address of a load-balanced MetaFrame server that will host a published application represented by an App object. The *clientName* is used for load balancing and reconnecting to disconnected sessions. For more information about initializing the CitrixWireGateway, see "CitrixWireGateway" on page 36.

public String getIPv4PortAlternateAddress(CitrixwireGateway *wireGateway*, String *clientName*)

Parameters

wireGateway

The CitrixwireGateway object from which to retrieve the server address.

clientName

The client name for load balancing and reconnecting disconnected sessions.

Return

Returns a string value that represents the alternate ipv4:port form of the address of a load-balanced MetaFrame server that will host a published

application represented by an App object. The *clientName* is used for load balancing and reconnecting to disconnected sessions. For more information about initializing the CitrixWireGateway, see “CitrixWireGateway” on page 36.

public String getDNSAddress(CitrixwireGateway *wireGateway*, String *clientName*)

Parameters

wireGateway

The CitrixwireGateway object from which to retrieve the server address.

clientName

The client name for load balancing and reconnecting disconnected sessions.

Return

Returns a string value that represents the DNS name of a load-balanced MetaFrame server that will host a published application represented by an App object. The *clientName* is used for load balancing and reconnecting to disconnected sessions. For more information about initializing the CitrixWireGateway, see “CitrixWireGateway” on page 36.

public String getDNSPortAddress(CitrixwireGateway *wireGateway*, String *clientName*)

Parameters

wireGateway

The CitrixwireGateway object from which to retrieve the server address.

clientName

The client name for load balancing and reconnecting disconnected sessions.

Return

Returns a string value that represents the DNS:port form name of a load-balanced MetaFrame server that will host a published application represented by an App object. The *clientName* is used for load balancing and reconnecting to disconnected sessions. For more information about initializing the CitrixWireGateway, see “CitrixWireGateway” on page 36.

public String getAddressResolutionType()

Returns the string value of the AddressResolutionType parameter in the NFuse.conf file.

Parameters

None.

Return

Returns the value of AddressResolutionType, and can be IPv4, IPv4-port, dns, or dns-port. If AddressResolutionType is not found in NFuse.conf or is empty, IPv4-port is returned.

public String getForceLoginDomain()

Returns the string value of the ForceLoginDomain parameter in the NFuse.conf file.

Parameters

None.

Return

Returns the value of ForceLoginDomain. If ForceLoginDomain is not found in NFuse.conf or is empty, an empty string 0 bytes long will be returned.

public boolean getForceLoginDomain()

Determines the value of the ForceLoginDomain parameter in the NFuse.conf file.

Parameters

None.

Return

Returns the value of ForceLoginDomain. If ForceLoginDomain is not found in NFuse.conf or is empty, an empty string 0 bytes long will be returned.

public boolean AllowCustomizeSettings()

Determines value of the AllowCustomizeSettings parameter in the NFuse.conf file.

Parameters

None.

Return

Returns a string value based on the value of AllowCustomizeSettings. If the value is **On**, it returns **True**. If AllowCustomizeSettings is not found in NFuse.conf, has a value other than **On**, or is empty, the string value returned will be **False**.

public boolean AllowCustomizeWindowSize()

Determines value of the AllowCustomizeWindowSize parameter in the NFuse.conf file.

Parameters

None.

Return

Returns a string value based on the value of AllowCustomizeWindowSize. If the value is **On**, it returns **True**. If AllowCustomizeWindowSize is not found in NFuse.conf, has a value other than **On**, or is empty, the string value returned will be **False**.

public boolean AllowCustomizeWindowColor()

Determines value of the AllowCustomizeWindowColor parameter in the NFuse.conf file.

Parameters

None.

Return

Returns a string value based on the value of AllowCustomizeWindowColor. If the value is **On**, it returns **True**. If AllowCustomizeWindowColor is not found in NFuse.conf, has a value other than **On**, or is empty, the string value returned will be **False**.

public boolean AllowCustomizeAudio()

Determines value of the AllowCustomizeAudio parameter in the NFuse.conf file.

Parameters

None.

Return

Returns a string value based on the value of AllowCustomizeAudio. If the value is **On**, it returns **True**. If AllowCustomizeAudio is not found in NFuse.conf, has a value other than **On**, or is empty, the string value returned will be **False**.

public boolean AllowCustomizeEncryption()

Determines value of the AllowCustomizeEncryption parameter in the NFuse.conf file.

Parameters

None.

Return

Returns a string value based on the value of AllowCustomizeEncryption. If the value is **On**, it returns **True**. If AllowCustomizeEncryption is not found in NFuse.conf, has a value other than **On**, or is empty, the string value returned will be **False**.

boolean allowLoginType(String loginType)

Tests whether the specified login type is permitted.

Parameters

loginType

A string specifying the login type to be checked. The values can be **Explicit**, **Guest**, **Certificate**, and **Integrated**.

Result

If the specified login type or types are permitted, the result is **True**. Otherwise, the result is **False**.

boolean onlyLoginType(String loginType)

Tests whether the specified login type is the only one permitted.

Parameters

loginType

A string specifying the login type to be checked. The value can be **Explicit**, **Guest**, **Certificate**, or **Integrated**.

Result

If the specified login type is the only one permitted, the result is **True**. Otherwise, the result is **False**.

int loginTypeCount()

This method returns the number of different authentication methods configured using the **AuthenticationMethods** setting in the NFuse.conf file.

Parameters

None.

Result

Returns the number of different authentication methods that are used, and can be a number between 0 and 4.

String allowUserChangePassword()

The value returned by this method will indicate under which circumstances the current user is allowed to change their password. Checking of this value should NOT be case-sensitive. A value of **Never** is assumed if an unexpected value is returned.

Return

Never: the user is not allowed to change their password.

Always: the user is allowed to change their password at any time.

Expired: the user is allowed to change an expired password only after it has expired.

Example Usage: App Object

To retrieve a property of an application, you must first create an App object. Next, call one of App's methods for the property you want to query.

The following Active Server Page/VBScript example illustrates how to query the Web server for the URL of an application's icon (.Gif file) stored on the server. Next, the example describes how to query a server farm for the application's description. The example code writes both of these pieces of information to a Web page.

In this example, myEnumerator is a previously created AppEnumerator object.

```
    'create an App object
set myApp = myEnumerator.nextAppFlat("")
    'write an HTML IMG tag that specifies the URL of the icon
    'file on the Web server
Response.Write "<img src='" & myApp.getIconfile & "'>"
    'write the application's description in the Web page
Response.Write myApp.getDescription & "<br>"
```

AppSettings

An AppSettings object is a container that holds the settable properties of a single application published in a MetaFrame server farm. Unlike an App object, which allows you to query application properties, an AppSettings object lets you modify application properties. At any given time, an AppSettings object can have none, some, or all of the settable application properties specified for an application.

Note A single AppSettings object can be applied to multiple App objects. Conversely, multiple App objects can draw from the settings specified in a single AppSettings object.

In addition to modifying an application's current properties, you can use an AppSettings object to handle application properties retrieved from a number of sources including an ODBC database or an "application settings" Web page. An AppSettings object could exist for each of these sources and be applied in turn to a given App object.

For the settings specified in an AppSettings object to actually be applied to an application, you must call App.applySettings on an App object for that application (see the description of the App object's applySettings() method on page 59 for more information).

AppSettings objects have the following methods:

void setWindowType(String *windowType*)

Sets the window type of the ICA session window for the referenced application.

Parameters

windowType

pixels: A size in pixels (when specifying size in pixels, you must also call the setWindowPixels() method).

percent: A size as a percentage of the client desktop (when specifying size as a percentage, you must also call the setWindowPercentage() method).

seamless: Seamless window.

fullscreen: Full screen.

Return

None.

void setWindowPixels(int *windowWidth*, int *windowHeight*)

Sets the width and height of the ICA session window for the referenced application. Specify values in pixels.

Parameters

windowWidth

The width of the client window in pixels. Value must be an integer.

windowHeight

The height of the client window in pixels. Value must be an integer.

Return

None.

void setWindowPercentage(int *windowScale*)

Specifies what percentage of the client desktop the ICA session will occupy.

Parameters

windowScale

The percentage of the client desktop. The value must be an integer between 0 and 100.

Return

None.

void setColorDepth(int *colorDepth*)

Specifies the number of colors used in the ICA session window to display the referenced application.

Parameters

colorDepth

1: 16 colors

2: 256 colors

4: High colors

8: True colors

Return

None.

void setSoundType(String *soundType*)

Specifies a level of sound support to apply to the referenced application.

Parameters

soundType

none: No sound support

basic: Sound 1.0

Return

None

void setEncryptionLevel(String *encryptionLevel*)

Specifies a level of encryption to use for the referenced application. To enable encryption levels higher than Basic, the MetaFrame server must support RC5 encryption. Support for RC5 encryption is included in MetaFrame XP, Feature Release 1 for MetaFrame 1.8, and SecureICA Services. MetaFrame for UNIX Operating Systems servers do not support RC5 encryption.

Parameters

encryptionLevel

basic: Basic encryption (XOR)

rc5-login: 128-bit for login only

rc5-40: 40-bit

rc5-56: 56-bit

rc5-128: 128-bit

Return

None.

void setVideoType(String *videoType*)

Specifies a level of video support to use for the referenced application. Use of Video 1.0 requires Citrix VideoFrame.

Parameters

videoType

none: No video

basic: Video 1.0

Return

None.

void setStartmenu(boolean *place*)

Allows you to set the Windows Start menu shortcut creation property to **On** or **Off** for the referenced application. This method does not allow you to actually place shortcuts on client devices.

Parameters

place

True: Set the referenced application's Start menu shortcut creation property to on.

False: Set the referenced application's Start menu shortcut creation property to off.

Return

None.

void setDesktop(boolean *place*)

Allows you to set the Windows desktop shortcut creation property to **On** or **Off** for the referenced application. This method does not allow you to actually place shortcuts on client devices.

Parameters

place

True: Set the referenced application's desktop shortcut creation property to on.

False: Set the referenced application's desktop shortcut creation property to off.

Return

None.

void setDescription(String *description*)

Specifies a description string for the referenced application.

Parameters

description

The description string you want to appear on the Web page.

Return

None.

void setFolder(String *folder*)

Specifies the folder in which the referenced application should be displayed.

Parameters

folder

The folder in which you want the application to appear.

Return

None.

void setIconFile(String *iconFile*)

Specifies the URL of an icon file (.Gif) for the referenced application. Use this method to override the referenced application's default icon file URL. See the description of the App object's getIconFile method on page 58 for more information about icon files.

Parameters

iconFile

The location of the icon file (.Gif) file relative to the Web server's root directory. When using this method, specify a string that can be placed to the right of SRC in the HTML tag <img SRC="path">.

Return

None.

String getLastError()

If the user of an AppSettings object detects that an error has occurred, getLastError() can be called to return a description of the error.

Parameters

None.

Return

A description of the last error on this object.

Example Usage: AppSettings Object

To override an application's settings, you must first create an AppSettings object. Next, call one of AppSetting's methods for the property you want to change and specify as its parameter the new value you want to apply to the application. To apply the new setting, you must call the App object's applySettings method with the AppSettings object specified as the parameter.

The following Active Server Page/VBScript example illustrates overriding an application's current description. In this example, myApp is a previously created App object.

```
    'create an AppSettings object
Set NewSettings = Server.CreateObject("com.citrix.nfuse.AppSettings")
    'call the setDescription method on the AppSettings object
NewSettings.setDescription("Application Description")
    'call the App object's applySettings method with
    'the parameter NewSettings
myApp.applySettings(NewSettings)
```

AppDataList

An AppDataList object contains a list of App objects. AppDataList objects are returned by a CitrixWireGateway object. You can save the AppDataList object in an AppListCache object for referencing by other sessions on the Web server.

You can use an AppDataList object to cache application set lists on the NFuse Classic Web server. When users log into the server, you can retrieve their application set information from the AppDataList quickly instead of querying the MetaFrame server repeatedly for information that may change infrequently.

AppDataList objects have the following methods:

AppEnumerator getAppEnumerator()

Retrieves an AppEnumerator object for an AppDataList object. You can then use the returned AppEnumerator object to enumerate the App objects in the AppDataList object.

Parameters

None.

Return

An AppEnumerator object or null if there was an error in communication with the MetaFrame server.

boolean isExpired()

Determines whether or not the AppDataList object is expired. The AppDataList timeout value (in seconds) is specified in the CacheExpireTime entry in NFuse.conf. You can call setExpireTime() to set the timeout value for the AppDataList object.

Parameters

None.

Return

True: The AppDataList object has expired.

False: The AppDataList object has not expired.

void addAppList(AppDataList *apps*)

Combines multiple AppDataList objects into a single object. Citrix recommends that only AppDataList objects from the same server farm be merged.

Parameters

apps

An AppDataList object to be merged into this AppDataList object.

Return

None

void setExpireTime(long t)

Specifies the timeout value for the AppDataList object. After this time period the object is no longer valid.

Parameters

t

Timeout value in milliseconds for this AppDataList object. Specify a negative number to cause the object to never expire.

Return

None.

long getCreateDate()

Returns the creation date of the object in milliseconds since January 1, 1970, 00:00:00 GMT.

Parameters

None.

Return

The creation date of the object in milliseconds.

long getExpireTime()

Returns the AppDataList object's expiration timeout value in milliseconds.

Parameters

None.

Return

The timeout value for this AppDataList object.

App findAppByExtension(String *fileName*)

Searches for a published application that is configured to support the file type impled by its file extension. For example, a Word document typically has the extension **.doc** and a published application (Word in this case) can be configured to support content with a file extension of **.doc**.

Parameters

filename

This specifies the address of the content (URL or path to a shared folder) and must include the file extension, such as **\\share\document.doc**.

Result

An App object coresponding to the published application supporting the specified file type.

Example Usage: AppDataList Object

The following Active Server Page/VBScript example illustrates how to create an AppDataList object. The example then uses the isExpired() method to determine if the AppDataList object is expired. In this example, myGateway is a previously created CitrixWireGateway object.

```
    'create an AppDataList object
Set myApps = myGateway.getAppDataList()
    'determine if the object is expired
expired = myApps.isExpired()
```

AppListCache

An AppListCache object caches AppDataList objects on your Web server so that you can reference them later to quickly retrieve application set information without contacting the server farm. To give all Web server sessions access to an AppListCache object, create the AppListCache object in the Web server's application scope. The AppListCache object saves data as key/AppDataList pairs. Each AppDataList object must have a unique name.

AppListCache objects have the following methods:

void addToCache(String *key*, Object *object*)

Adds an AppDataList object to the AppListCache object.

Parameters

key

The key by which to identify the AppDataList object.

object

The name of the AppDataList object to add.

Return

None.

AppDataList removeFromCache(String *key*)

Removes an AppDataList object from the AppListCache object.

Parameters

key

The key associated with the AppDataList object to remove.

Return

This method returns the object to be removed. It returns null if the object could not be found.

boolean contain(String *key*)

Determines whether or not the AppListCache object contains a specific AppDataList object.

Parameters

key

The key associated with a specific AppDataList object.

Return

True: The AppListCache object contains the specified AppDataList object.

False: The AppListCache object does not contain the specified AppDataList object.

AppDataList retrieveFromCache(String *key*)

Retrieves the specified AppDataList object from the AppListCache object.

Parameters

key

The key associated with the AppDataList object to retrieve.

Return

The specified AppDataList object or null if the object was not found.

int size()

Retrieves the number of AppDataList objects stored in the AppListCache object.

Parameters

None.

Return

An integer representing the number of AppDataList objects contained in the AppListCache object.

void compactCache()

Removes expired AppDataList objects from this AppListCache object.

Parameters

None.

Return

None.

void setOptimalNumber(int *totalNumber*)

Specifies the optimal number of objects to store in the AppListCache object. The addToCache() method calls compactCache() automatically if the total number of objects stored in this AppListCache object has reached this number.

The AppListCache object does not enforce the maximum number of objects that can be stored. It is possible to have more than this number of objects stored even after compactCache() is called.

Parameters

totalNumber

The optimal number of objects to store in this AppListCache object. When an AppListCache object is created, this optimal number is set to 100 by default.

Return

None.

Example Usage: AppListCache Object

The following Active Server Page/VBScript example illustrates how to create an AppListCache object. The example then uses the size() method to

determine the number of AppDataList objects the AppListCache object contains.

```
    'create an AppListCache object
Set myCitrixCache = Server.CreateObject("com.citrix.nfuse.AppListCache")
    'determine if the object is expired
size = myCitrixCache.size()
```

TemplateParser

A TemplateParser object performs symbol substitution on a template text file. During processing, a TemplateParser object searches for and replaces Citrix substitution tags. One use of a TemplateParser object involves calling the object from a Web interface to take a template ICA file and replace all the application-specific parts of the file with the appropriate properties of an App object.

For a list of Citrix substitution tags, see Chapter 2, "Using Citrix SubstitutionTags" on page 13.

TemplateParser objects have the following methods:

void setCookieSessionFields(String *cookie*)

Sets the session fields of the TemplateParser object to the values specified in the passed-in cookie string. Whether a given session field is set or not depends on its current state and the override order specified in the SessionFieldLocations entry in the configuration file NFuse.conf.

Parameters

cookie

A string representing the cookies passed to the Web server in the current HTTP request, in the form "Name1=Value1&Name2=Value2&…." or "&Name1=Value1&Name2=Value2&…."

Return

None.

void setSingleCookieSessionField(String *sessionField*, String *value*)

Sets a single session field of the TemplateParser object to the value specified. Whether a given session field is set or not depends on its current state and the override order specified in the SessionFieldLocations entry in the configuration file NFuse.conf.

Parameters

sessionField

The name of the session field to set.

value

The value to which to set the session field.

Return

None.

void setUrlSessionFields(String *url*)

Sets the session fields of the TemplateParser object to the values specified in the passed-in URL query string. Whether a given session field is set or not depends on its current state and the override order specified in the SessionFieldLocations entry in the configuration file NFuse.conf.

Parameters

url

A string representing the URL query string received by the Web server in the current HTTP request, in the form "Name1=Value1&Name2=Value2&…." or "&Name1=Value1&Name2=Value2&…."

Return

None.

void setSingleUrlSessionField(String *sessionField*, String *value*)

Sets a single session field of the TemplateParser object to the value specified. Whether a given session field is set or not depends on its current state and the override order specified in the SessionFieldLocations entry in the configuration file NFuse.conf.

Parameters

sessionField

The name of the session field to set.

value

The value to which to set the session field.

Return

None.

void setPostSessionFields(String *post*)

Sets the session fields of the TemplateParser object to the values specified in the passed-in HTTP Post. Whether a given session field is set or not depends on its current state and the override order specified in the SessionFieldLocations entry in the configuration file NFuse.conf.

Parameters

post

A string representing the Post received by the Web server in the current HTTP request, in the form "Name1=Value1&Name2=Value2&…." or "&Name1=Value1&Name2=Value2&…."

Return

None.

void setSinglePostSessionField(String *sessionField*, String *value*)

Sets a single session field of the TemplateParser object to the value specified in the passed-in HTTP Post. Whether a given session field is set or not depends on its current state and the override order specified in the SessionFieldLocations entry in the configuration file NFuse.conf.

Parameters

sessionField

The name of the session field to set.

value

The value to which to set the session field.

Return

None.

void setSessionFields(String *input*)

Sets the values of session fields from a Web server (script file) interface. Whether a given session field is set or not depends on its current state and the override order specified in the SessionFieldLocations entry in the configuration file NFuse.config.

Parameters

input

A string representing the session fields to set, in the form "Name1=Value1&Name2=Value2&…." or "&Name1=Value1&Name2=Value2&…."

Return

None.

void setSingleSessionField(String *sessionField*, String *value*)

Sets the value of a single session field from a Web server (script file) interface. Whether the session field is actually set or not depends on its current state and the override order specified in the SessionFieldLocations entry in the configuration file NFuse.config.

Parameters

sessionField

The name of the session field to set.

value

The value to which to set the session field.

Return

None.

boolean Parse()

Parses a template text file. The template to parse must be specified in a session field (NFuse_Template). The result of the parsing (with the substitution tags processed and replaced) is accessed using the getNextDataBlock() method.

Parameters

None.

Return

True: The parsing succeeded. You can now call getNextDataBlock() to get the result of the parse.

False: The parsing failed. Do not call getNextDataBlock(). Call getLastError() to get the error.

String getNextDataBlock()

Returns a portion of the parsed template. To fully retrieve the parsed template, call this method repeatedly until it returns the empty string.

Parameters

None.

Return

A string of some maximum length. If there is no more parsed template to return, this method returns the empty string (“”).

String getContentType()

Returns the MIME "Content-Type" for the result of this template parse. For instance, when parsing a template for an ICA file, this might be "application/x-ica."

Parameters

None.

Return

The MIME type.

String getLastError()

If the user of the TemplateParser object detects that an error has occurred, getLastError() can be called to return a description of the error.

Parameters

None.

Return

A description of the last error on this object.

void setClientIPv4Address(String *clientAddress*)

Sets the client address in IPv4 format.

Parameter

clientAddress
The client address to be set, in IPv4 format.

Return

None.

void setClientAddress(String *clientAddress*)

Sets the user's client address.

Parameter

clientAddress
The client address to be set.

Return

None.

void setClientAddressType(String *clientAddress*)

Sets the user's client address type.

Parameter

clientAddress

The client address type to be set. Valid values include **DNS, DNS-PORT, ipv4, ipv4-port, and URI.**

Return

None.

Example Usage: TemplateParser Object

The following Active Server Page/VBScript example illustrates how to create a TemplateParser object. The example then retrieves a cookie containing parameter/value pairs, where the parameters are written in the syntax of NFuse Classic session fields. Finally, the example sets the session fields retrieved from the cookie so that they are available to the TemplateParser as it parses an NFuse Classic template. In this example, NFuseData is a previously created cookie.

```
    'create a TemplateParser object
Set myParser = Server.CreateObject("com.citrix.nfuse.TemplateParser")
    'retrieve a cookie containing session field/value pairs
CookStr = Request.Cookies("NFuseData")
    'set the session fields contained in the cookie
myParser.setCookieSessionFields(CookStr)
```

PropertiesKeys

A PropertiesKeys object retrieves the properties in the NFuse Classic configuration file. In addition, it is used to get a static text string from the provided text file. This is done to provide a way to implement language-independent ASP and JSP code.

This object also provides scripts with access to the content of the NFuse.conf file and to localized strings. This allows the adminstrator interface to dynamically reload the current contents of the NFuse.conf file, for example, after it has been modified by the NFuse Classic Administration Tool.

PropertiesKeys has the following methods:

String getProperty(String *name*)

Returns the property value for this property name.

Parameter

name

A string representing the property name.

Return

Returns null if the property does not exist.

String getProperty(String *key*)

Gets the value of the specified property key from the NFuse.conf file.

Parameter

key

A string representing the property key, such as **AuthenticationMethods**.

Return

The value of the setting specifed by the key as a string.

boolean containsProperty(String *key*)

Tests if the specified property exists.

Parameter

key

A key by which to identify a specified property.

Return

Returns **True** if the NFuse Classic configuration file contains the specified property.

String getStaticString(String *key*)

Get the localized string corresponding to the specified key.

Parameter

key

The name of the string property for which the localized value is required.

Return

This returns an empty string if the specified static string does not exist.

String getStaticString(String *key*, String *arg1*)

Parameter

key

A key by which to identify the static string.

Return

Returns the static string from the global string table, as above. In addition, it replaces the placeholder (%s) in the static string with arg1.

String getStaticString(String *key*, String *arg1*, String *arg2*)

Parameter

key

A key by which to identify the static string.

Return

Returns the static string from the global string table and replaces the placeholder (%s) with *arg1* and *arg2*.

void reloadProperties()

Dynamically reload the contents of the NFuse configuration file.

Parameters

None.

Return

Non.

AnonCredentials

The AnonCredentials object is passed to the CitrixWireGateway object to allow it to retrieve anonymous (guest) applications.

AnonCredentials()

Instantiates the AnonCredentials object. No initialization of this object is necessary. Pass the instantiated object as a *credential* argument to an instance of the CitrixWireGateway object.

Parameters

None.

AuthenticatedCredentials

An AuthenticatedCredentials object is a container that holds the details of a user account that has been autheticated using an external authentication mechanism. For example, a user logged in a using a Smart Card will be authenticated by the system using the Certificate contained in the Smart Card. NFuse users will be authenticated by the web server (IIS).

AuthenticatedCredentials holds the user account identifier and a list of associated secure identifiers (SIDs) and has several methods, listed below.

void initialize(String *identifier*)

Intializes an AuthenticatedCredentials object with its associated account identifier.

Parameter

identifier

This can be an NT 4 System Account Manager (SAM)-style identifier, for example *domain**name* or an Active Directory Services (ADS)-style/UPN identifier, such as *name*@*domain*.

Return

None.

void addSid(String *sid*)

Adds the specified secure identifier to the list assocated with the user account represented by the AuthenticatedCredentials object.

Parameter

sid

The secure identifier to add to the user account.

Return

None.

String getIdentifier()

Returns the identifier supplied by the initialize call.

Parameters

None.

Return

The identifier supplied by the initialize call.

String getIdentifierType()

Returns the type of identifier as a string.

Parameters

None.

Return

This returns **SAM** for NT4 SAM-style identifiers and **UPN** for ADS-style identifiers.

void setIdentifierType(String *identifierType*)

Sets the type of identifier to **SAM** or **UPN**.

Parameter

identifierType
AuthenticatedCredentials.SAM or **AuthenticatedCredentials.UPN**

Return

None.

String[ ] getSIDS()

Return the list of SIDs as an array of strings.

Parameters

None.

Return

Returns the list of SIDs as an array of strings.

CHAPTER 4

ICA File Reference

The template ICA files included in NFuse Classic allow you to use a single template as the basis for delivering a customized ICA file to each user for each requested application. For example, if you have a group of users who always require ICA sessions of a certain resolution, you can use window size parameters to hard-code that resolution into the template ICA file instead of using a substitution tag to query the MetaFrame server for window size information. Or, use client device mapping parameters to enable or disable client devices such as printers and COM ports.

The presence of substitution tags in the template file makes this customization possible. When creating an ICA file from the template, the Java objects replace each substitution tag with the corresponding information about the user or desired application.

In addition to using NFuse Classic's substitution tags to customize a template file, you can also directly edit a template by hard-coding ICA file parameters. Hard-coded ICA file parameters give you access to various ICA session properties not configurable using substitution tags or standard application publishing.

This chapter contains the following:

- Information about the structure and contents of ICA files
- Information about using ICA files with firewalls
- Information about using embedded ICA clients

ICA File Structure

An *ICA file* is a text file containing information about a published application. ICA files are written in Ini file format and organize published application information in a standardized way that ICA Clients can interpret. The following example depicts basic ICA file layout:

```
<[NFuse_setSessionField NFuse_ContentType=application/x-ica]>
[WFClient]
Version=2

[ApplicationServers]
ApplicationName=

[ApplicationName]
Parameter1=Value
Parameter2=Value
Parameter3=Value
```

[WFClient]

The [WFClient] section is the first section in an ICA file and must contain at least the parameter/value pair Version=2. The version number is for Citrix internal use and should not be modified.

In addition to the version parameter, the template ICA file included in Chapter 2 specifies a second parameter/value pair. NFuse Classic uses this pair (ClientName=[NFuse_ClientName]) to place a generated, unique ICA Client name in the ICA file. This client name identifies the ICA Client connecting to the published application and is a requirement of various ICA Client functions including printing and session reconnect.

Note NFuse Classic generates the client name for ICA connections from the username and domain of the user, and is visible in the Citrix Management Console. However, this may appear in the Console in a shortened form and may contain characters not present in the original username or domain. This ensures unique client names and does not indicate a problem.

The template ICA file in Chapter 2 contains a single entry previous to the [WFClient] section. This first entry is a session field setting command used by NFuse Classic to set the MIME type of the ICA file to application/x-ica. This entry is required for NFuse, but is not a requirement in standard ICA files.

[ApplicationServers]

The [ApplicationServers] section contains a single parameter. This parameter specifies the name of a Citrix published application. Following the published application name is an equals sign (=).

For example, in an ICA file for a published application named “Notepad,” the [ApplicationServers] section contains the following:

Notepad=

In the template ICA file, use a substitution tag to enter the published application name in the [ApplicationServers] section; for example:

[NFuse_AppName]=

[*ApplicationName*]

The final required section in an ICA file is [*ApplicationName*], where *ApplicationName* is the name of the published application specified in the [ApplicationServers] section. In the template ICA file, the published application name can be specified using the substitution tag [NFuse_AppName]; for example, [[NFuse_AppName]].

The [*ApplicationName*] section contains configuration information for the specified published application. This information is in the form of a parameter/value pair list. The following topics describe some parameters you can use to customize your template ICA files.

ICA File Parameters

The following topics list parameters that configure:

- General ICA session properties
- User credentials
- Window size and number of colors
- Persistent bitmap caching
- ICA Client TCP/IP browsing
- Encryption
- SOCKS proxy servers

- SpeedScreen Latency Reduction
- Client Auto Update
- Client device mapping

Important Not all parameters are supported by all ICA Clients or all MetaFrame server products.

General Parameters

The following parameters configure basic ICA session properties.

InitialProgram	The name of the published application prefixed with a pound sign (#); for example, for a published application named "Notepad," specify: InitialProgram=#Notepad
WorkDirectory	Specifies a working directory for the application to use.
TransportDriver	Transport protocol used to connect to the MetaFrame server. Always set to TCP/IP.
WinStationDriver	Version of the ICA protocol to use for the connection. Always set to ICA 3.0.
MouseTimer	Specifies a time interval in milliseconds during which mouse input is collected before being sent to the MetaFrame server. The default value of 100 milliseconds is optimized for WANs. In a Dial-In or LAN environment, reducing this value can give better responsiveness. Using too low a value in a LAN environment can generate a large number of small packets, which can affect network performance.
KeyboardTimer	Specifies a time interval in milliseconds during which keyboard input is collected before being sent to the MetaFrame server. The default value of 100 milliseconds is optimized for WANs. In a Dial-In or LAN environment, reducing this value can give better responsiveness. Using too low a value in a LAN environment can generate a large number of small packets, which can affect network performance.
SwapButtons	Specifies whether or not to switch the function of the client device's left and right mouse buttons. Specify Yes to switch button functions.

WindowsCache	Specifies the size of the ICA Client's memory cache. The default for the ICA Win32 Client is 3.5MB.
ICAPortNumber	By default, MetaFrame servers and ICA Clients use TCP/IP port 1494 to pass ICA traffic. Add this parameter to cause the ICA Client to use some other TCP/IP port. To use this parameter, you must also configure the MetaFrame server to use a non-default port. If your server is a MetaFrame for Windows server, see your server documentation for information about using the ICAPORT utility. For MetaFrame for UNIX servers, see your server documentation for information about using the CTXCFG utility. If your ICA Client does not support the ICAPORT parameter, you can specify the desired port by appending :*port#* to the ICA file's address parameter; for example, to use port 80: Address=[NFuse_IPv4Address]:80
NFuse_AppServerAddress	Retrieves the address of the MetaFrame server hosting the published application. The format of the address is specified in the NFuse.conf file. The parameters of this are **on**, **off**, and **mapped** to allow client and server address mapping.
NFuse_IcaAudio	Retrieves the ICA session audio information for placement in ICA files. The TemplateParser object replaces this tag with all the audio information required in an ICA file.
NFuse_sessionSharingKey	Retrieves the color depth, encryption, audio quality, domain name, user name and farm name of the client. If this information is the same for different sessions on the same client device, the client will attempt to share the session.
NFuse_ipv4Address_Port	Returns the ipv4:port address of the application's resolved, load-balanced server.
NFuse_ipv4AddressAlternate_Port	Returns the alternate ipv4:port address of the application's resolved, load-balanced server. The alternate address is intended to be exposed to clients outside of the firewall. This is set in the ICA Browser using the **altaddr** utility.
NFuse_DnsAddress	Returns the DNS name of the application's resolved, load-balanced server.

NFuse_DnsAddressAlternate	Returns the alternate DNS name of the application's resolved, load-balanced server.
NFuse_DnsAddress_Port	Returns the DNS name:port of the application's resolved, load-balanced server.
NFuse_DnsAddressAlternate _Port	Returns the alternate DNS name:port of the application's resolved, load-balanced server.

User Credential Parameters

User credential parameters identify the user attempting to connect to the published application.

Username	A user name supported by your MetaFrame server's account authority; for example, a Windows NT user name if your MetaFrame server is a Windows Terminal Server or Windows 2000 Server Family system.
Domain	A Windows NT domain name.
Password	A valid password for the specified user account. ICA Clients expect password information entered for the Password parameter to be in scrambled format. Use the [NFuse_PasswordScrambled] substitution tag to enter passwords in scrambled format.
ClearPassword	Used to specify a password in clear text. In some cases, to use a clear text password, the Password field must also be included in the ICA file and set to a null value; for example: Password=
AutoLogonAllowed	By default, ICA connections that use encryption levels greater than Basic do not accept the user name, domain, and password information specified in an ICA file. Such connections force users to manually log in to each application they connect to, regardless of whether the ICA file already contains their credentials. Use this parameter to force the server to accept user credentials specified in an ICA file. For an example of this tag's usage, see "Setting Encryption Parameters" on page 106. Specify On to allow automatic logon. This is not required for connections between servers running MetaFrame XP 1.0 for Windows at the Feature Release 2 level with the latest ICA Client.

Window Size and Color Parameters

Window size and color parameters control display properties of the ICA session window.

DesiredColor	Number of colors used to display the ICA session window. 1=16 colors 2=256 colors 4=High Color (16-bit) 8=True Color (24-bit)
DesiredHRES	Specifies the width of the ICA session window in pixels; for example, 640.
DesiredVRES	Specifies the height of the ICA session window in pixels; for example, 480.
ScreenPercent	Specifies the horizontal and vertical pixel resolution as a percentage of the client desktop. If the ScreenPercent field is present, DesiredHRES and DesiredVRES fields are ignored.
TWIMode	If seamless ICA session windows are supported, this parameter causes the ICA Client to create a seamless ICA session window. Seamless session windows place the published application in a resizable window. Specify On for a seamless connection. In some cases, when enabling a seamless connection you must additionally specify the following two parameter/value pairs: DesiredHRES=0xffffffff DesiredVRES=0xffffffff

DesiredWinType	Specifies desired window type for a custom Program Neighborhood connection. 1=640x480 2=800x600 3=1024x768 4=1280x1024 5=Custom 6=Percent 7=Full Screen 8=Seamless 0=None. The connection inherits the default setting from the ICA Client.

Persistent Caching Parameters

Persistent caching parameters control the storing of commonly-used graphical objects such as bitmaps in a local cache on the client device's hard disk.

PersistentCacheEnabled	Enables and disables persistent bitmap caching. Specify On to enable caching.
PersistentCacheSize	Specifies the amount of disk space in bytes to use for bitmap caching.
PersistentCacheMinBitmap	Specifies the smallest bitmap in bytes that can be cached to disk.
PersistentCachePath	Specifies the location of the directory containing the cached image data; for example: D:\WTSRV\Profiles\User1\Application Data \ICAClient\Cache

TCP/IP Browsing Parameters

TCP/IP browsing parameters configure ICA Client server location. Server location provides a method for ICA Clients to resolve published application names into MetaFrame server IP addresses.

TcpBrowserAddress	Specifies the IP address of a MetaFrame server used for server location and published application name resolution. Specify up to 15 TCP browser addresses by entering: TcpBrowserAddress2=*x.x.x.x* TcpBrowserAddress3=*x.x.x.x*
BrowserTimeout	Specifies the number of milliseconds the ICA Client waits for a response after making a request to the master browser. The master browser request is an initial step required by server location and published application name resolution. This setting is useful in environments where the ICA Client's master browser request must pass through various impediments to quick response such as a WAN connection.
BrowserRetry	Specifies the number of times an ICA Client resubmits a master browser request that has timed out. The master browser request is an initial step required by server location and published application name resolution.
UseAlternateAddress	Defines whether to use a server's alternate address for ICA connectivity across a firewall or a router. Specify 1 to cause the ICA Client to use the MetaFrame server's alternate address. To use this parameter, you must also configure the MetaFrame server. If your server is a MetaFrame for Windows server, see your server documentation for information about using the ALTADDR utility. For MetaFrame for UNIX servers, see your server documentation for information about using the ctxalt utility.

Setting NFuse Classic Server Location Options

NFuse Classic's substitution tags allow you several methods of specifying a published application name/address in a template ICA file. The method you use depends upon where in the NFuse Classic system you want to perform MetaFrame server location.

The following three options explain which substitution tag to use in your template ICA files to correctly resolve published application names:

- **Web-server-side server location**. Some network configurations using routers and multiple subnets may require you to specify the published application name as the IP address of the MetaFrame server hosting the application. Specifying the name as an address in the ICA file forces the Java objects on the Web server to resolve the published application name instead of relying upon the ICA Client to perform the translation. The Java objects, unlike the ICA Clients, do not use an initial UDP broadcast to locate a MetaFrame server and therefore eliminate the multiple subnet complications of ICA-Client-to-Citrix-server UDP broadcasting. (The Citrix-server-to-Citrix-server communication that occurs after the initial server location is, however, UDP-based.)

  To perform Web-server-side server location, specify [NFuse_IPv4Address] as the published application address in your template ICA files; for example:

  ```
  [ApplicationServers]
  [NFuse_AppName]=

  [[NFuse_AppName]]
  Address=[NFuse_IPv4Address]
  ```

  When using the [NFuse_IPv4Address] tag, Citrix recommends that your ICA file be configured for client name. Please see the end of this topic for more information. The [NFuse_IPv4Address] tag is the default entry used in the NFuse Classic template ICA files.

- **ICA-Client-side server location**. This method causes the ICA Client to perform published application name resolution in the typical UDP-based fashion of ICA Clients. This method is useful if you want to place the address resolution load on individual ICA Clients instead of on the Java objects on the Web server.

To perform ICA-Client-side server location, specify [NFuse_AppName] as the published application address in your template ICA files; for example:

```
[ApplicationServers]
[NFuse_AppName]=

[[NFuse_AppName]]
Address=[NFuse_AppName]
```

In a multiple subnet environment, in which the ICA Client and MetaFrame server farm are on different subnets, you must also specify a TCPBrowserAddress entry so that the ICA Client can locate the master ICA Browser on the other subnet; for example:

```
[ApplicationServers]
[NFuse_AppName]=

[[NFuse_AppName]]
Address=[NFuse_AppName]
TCPBrowserAddress=x.x.x.x.
```

- **Server location through firewalls**. To perform published application name resolution across a firewall that uses network address translation, use the [NFuse_IPv4AddressAlternate] substitution tag to specify the published application name. This tag identifies the published application with the external server address of the MetaFrame server hosting the application; for example:

```
[ApplicationServers]
[NFuse_AppName]=

[[NFuse_AppName]]
Address=[NFuse_IPv4AddressAlternate]
```

Like Web-server-side server location, this method performs name resolution on the Web server. To use this method, your MetaFrame server must be configured using the ALTADDR (MetaFrame for Windows) or ctxalt (MetaFrame for UNIX) utility. See your server documentation for more information.

When performing Web-server-side server location and server location through firewalls, Citrix recommends that your template ICA file's [WFClient] section contain the following parameter/value pair: Clientname=[NFuse_ClientName]. Placing this pair in the ICA file causes the client name used by the Java objects during resolution and the client name used by the ICA Client for the ICA session

to be identical. This method works for all supported ICA Clients except the ICA Java Client.

When using the ICA Java Client to embed ICA sessions in a Web page, you must use the client name parameters of the Java Client's Applet tag to include client name information for the ICA session. For example, to use the Java Client's client name parameters in a substitution-tag-based site, the Applet tag must include the following:

```
<param name=client.wfclient.usehostname value='no'>
<param name=client.wfclient.clientname value='[NFuse_ClientName]'>
```

In a scripted site, your Applet tag might look like this:

```
<param name=client.wfclient.usehostname value='no'>
<param name=client.wfclient.clientname value='<%=app.generateClientName(credentials)%>'>
```

Setting Encryption Parameters

Encryption parameters configure the level of encryption to use when sending data between the ICA Client and MetaFrame server. To enable encryption levels higher than Basic, the MetaFrame server must support RC5 encryption. Support for RC5 encryption is included in MetaFrame XP, Feature Release 1 for MetaFrame 1.8, and SecureICA Services.

EncryptionLevelSession	Selects the level of encryption for the ICA connection. Possible values include: EncRC5-40=40-bit encryption EncRC5-56=56-bit encryption EncRC5-128=128-bit encryption EncRC5-0=128-bit encryption (login only) If this parameter is left unspecified, the ICA session uses the default level (Basic).

Note Support for RC5 encryption is not available if your server farm is composed of MetaFrame for UNIX Operating Systems servers.

When specifying an encryption level other than Basic, you must include in the ICA file an additional section that lists drivers the ICA Client must load to support the specified level of encryption. The section takes the name of the value specified for EncryptionLevelSession. For example, to use 56-bit encryption, create a section called [EncRC5-56]. In this section, list the driver to load for each ICA Client:

```
[EncRC5-56]
DriverNameWin32=PDC56N.DLL
DriverNameWin16=PDC56W.DLL
```

Configuring Authentication Over Encrypted Connections

In its most secure configuration, RC5 encryption uses 128-bit encryption to encrypt user authentication to the MetaFrame server. To establish this secure connection, the ICA Client and MetaFrame server must negotiate the connection prior to the ICA Client passing user credentials to the MetaFrame server for user log in. MetaFrame servers require that ICA connections using RC5 encryption allow automatic login for servers running MetaFrame XP 1.0 for Windows, Feature Relase 2, and the latest ICA Client.

The standard Template.ica file uses the NFuse_IcaEncryption substitution tag to conditionally include user credentials depending on the encryption level.

```
[NFuse_IcaEncryption]
```

The NFuse_IfSessionField and /NFuse_IfSessionField tags create a conditional block that causes the Java objects to execute the enclosed code only if a certain condition is met. When processing the example above, the Java objects place the user's credentials in the finished ICA file only if the session's encryption level is Basic. For all other levels, the Java objects omit all credential information.

In some deployment scenarios, you may want to simultaneously support automatic login (sending user credentials in the ICA file) and encryption levels greater than Basic. To do so, you must include the AutoLogonAllowed parameter in the section of the template ICA file that contains user credentials; for example:

```
[[NFuse_AppName]]
Address=[NFuse_IPV4Address]
InitialProgram=#[NFuse_AppName]
DesiredColor=[NFuse_WindowColors]
TransportDriver=TCP/IP
WinStationDriver=ICA 3.0
AutoLogonAllowed=On
Username=[NFuse_User]
Domain=[NFuse_Domain]
Password=[NFuse_PasswordScrambled]
[NFuse_IcaWindow]
```

Note When enabling automatic login, you do not need to specify the NFuse_IfSessionField and /NFuse_IfSessionField tags in the template ICA file.

Additionally, make sure your template ICA file contains a [Compress] section with all required compression drivers listed; for example:

```
[Compress]
DriverName=PDCOMP.DLL
DriverNameWin16=PDCOMPW.DLL
DriverNameWin32=PDCOMPN.DLL
```

Using Ticketing Over Encrypted Connections

Ticketing provides the most convenient and secure method of authentication by allowing automatic logon without explicitly placing user credentials in ICA files.

Note Support for ticketing is not available in the first release of the Citrix XML Service for MetaFrame for UNIX Operating Systems servers. Support for ticketing is available for servers running MetaFrame for UNIX Operating Systems, Feature Release 1.

When you create sites with ticketing enabled, the Java objects place an authentication ticket instead of actual user credentials in the ICA file sent to a client device. A user executing an ICA file containing a ticket automatically logs on to the application without entering actual credentials. Instead, the server farm stores the user's credentials within the farm and retrieves the credentials when presented with the ticket. The MetaFrame server then passes the actual credentials to the MetaFrame server that will host the desired application.

A template ICA file that supports ticketing can look like the following:

```
[[NFuse_AppName]]
Address=[NFuse_IPV4Address]
InitialProgram=#[NFuse_AppName]
DesiredColor=[NFuse_WindowColors]
TransportDriver=TCP/IP
WinStationDriver=ICA 3.0
[NFuse_Ticket]
[NFuse_IcaWindow]
```

To use ticketing with RC5 encryption, make sure your template ICA files contain the AutoLogonAllowed=On parameter.

Setting SOCKS Proxy Parameters

SOCKS parameters configure the ICA Client to work with SOCKS proxy servers. For information about SOCKS proxy servers, see the *Citrix ICA Client Administrator's Guide* for your ICA Client.

ICASOCKSProtocolVersion	Indicates which version of the SOCKS protocol to use for the connection. Possible values include: -1: None. Do not use SOCKS for this connection. 0: Autodetect. Client determines which version the proxy is using. 4: Use SOCKS version 4. 5: Use SOCKS version 5.
ICASOCKSProxyHost	Specifies the DNS name or IP address of the SOCKS proxy to use for this connection.
ICASOCKSProxyPortNumber	Port number of the SOCKS proxy server (usually 1080).
ICASOCKSProxy	Will generate SOCKS settings automatically if this tag is used

To use a SOCKS proxy server with NFuse Classic, you must add the above parameters to both the [WFClient] and [[NFuse_AppName]] sections of the template ICA file; for example:

```
<[NFuse_setSessionField NFuse_ContentType=application/x-ica]>
<[NFuse_setSessionField NFuse_WindowType=seamless]>

[WFClient]
Version=2
ClientName=[NFuse_ClientName]
ICASOCKSProtocolVersion=0
ICASOCKSProxyHost=someServerAddress
ICASOCKSProxyPortNumber=somePortNumber

[ApplicationServers]
[NFuse_AppName]=

[[NFuse_AppName]]
Address=[NFuse_IPV4Address]
InitialProgram=#[NFuse_AppName]
DesiredColor=[NFuse_WindowColors]
TransportDriver=TCP/IP
WinStationDriver=ICA 3.0
<[NFuse_IFSESSIONFIELD sessionfield="NFUSE_ENCRYPTIONLEVEL" value="basic"]>
Username=[NFuse_User]
Domain=[NFuse_Domain]
Password=[NFuse_PasswordScrambled]
<[/NFuse_IFSESSIONFIELD]>
<[NFuse_IFSESSIONFIELD sessionfield="NFUSE_SOUNDTYPE" value="basic"]>
ClientAudio=On
<[/NFuse_IFSESSIONFIELD]>
[NFuse_IcaWindow]

ICASOCKSProtocolVersion=0
ICASOCKSProxyHost=someServerAddress
ICASOCKSProxyPortNumber=somePortNumber
```

SpeedScreen Latency Reduction Parameters

The following parameters configure support for SpeedScreen's local text echo and mouse feedback features. On slow networks, ICA Client users can experience delays between an action such as keyboard input and the appearance, or echoing, of the text on screen. Similarly, a mouse click and visual confirmation of the click, such as the appearance of an hourglass, can also be separated by network latency. SpeedScreen enhances user experience by providing users with immediate keyboard and mouse feedback. For more information about SpeedScreen, including instructions on configuring your server's applications to use SpeedScreen features, see your server documentation.

ZLKeyboardMode	Specifies support for local text echoing. 0: Disable text echoing. 1: Enable text echoing. 2: Automatically enable or disable text echoing depending upon a check of the connection's latency. This is the default value.
ZLMouseMode	Specifies support for mouse feedback. 0: Disable mouse feedback. 1: Enable mouse feedback. 2: Automatically enable or disable mouse feedback depending upon a check of the connection's latency. This is the default value.

Client Auto Update Parameters

Client Auto Update parameters configure whether or not the ICA Client accepts client program updates from the MetaFrame server. See your server documentation for information about server-side configuration of Client Auto Update and a list of updatable ICA Clients.

UpdatesAllowed	Configures support for Auto Client Update. Specify On to allow updates or Off to disallow updates. The default value is On.

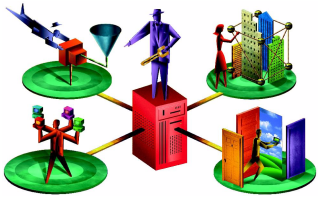
Client Device Mapping Parameters

Client device mapping parameters enable and disable client services such as sound support and client drive mapping.

COMAllowed	Enables or disables client COM port mapping. Specify Yes to enable, No to disable.
CPMAllowed	Enables or disables client printer mapping. Specify Yes to enable, No to disable.
CDMAllowed	Enables or disables client drive mapping. Specify Yes to enable, No to disable.
DisableSound	Enables or disables ICA Client sound support for simple beeps. Specify Off to disable sound. Leave out or specify On to enable.
ClientAudio	Like DisableSound, this parameter enables or disables ICA Client audio support. Specify Off to disable audio. Specify On to enable.
VSLAllowed	Enables or disables support for the Microsoft and Novell TCP stacks. Specify Yes to enable, No to disable.

CHAPTER 5

Example Scripts



This chapter contains example scripts that are designed to act as a guide to using NFuse Classic objects.

Note MetaFrame XP Setup offers administrators the option of installing NFuse Classic during MetaFrame installation. This option installs the Web Server Extension on the server and places the NFuse Classic Web site in the Web server's document root directory.

Introduction

The scripts in this chapter are provided to show how to use the objects in two environments:

1. Microsoft IIS - making use of VBscript and Active Server Pages
2. Linux and Apache Web Server - making use of Java Script Pages

How to Use the Scripts

To use this Software Development Kit, you must:

- Have NFuse Classic 1.7 set up on your server
- Publish applications in several different program neighborhood directories
- Set up a directory at the same level as the NFuse Classic Web pages on the webserver (such as *examples*)

Each example described in this chapter can be directly cut and pasted into a text file. Each example contains one or more files; all of the example files can be created in the same directory under the webserver.

1. Create a new .txt file
2. Cut and paste the example script into the .txt file
3. Replace *username*, *domain*, and *password* with valid strings
4. Save the .txt file as the filename in the example
5. Move the file to the examples directory
6. Execute the script through an Internet Browser; the start page is indicated in the example

All URLs are described with `%webpath%`, indicating the path to the example directory on the server being used to run these examples.

Note To select and copy text presented in a PDF file:

1. Activate the **Text Selection Tool** in Acrobat Reader.
2. Select the desired text.
3. Right-click the selected text, then select **Copy** from the pop-up menu.
4. Paste the text into your text editor.

For more information about using Acrobat Reader, see Acrobat's help files.

ASP Examples

Simple (Flat) Application Iteration

The examples in this section show all published applications available to the user in a flat structure, regardless of the Program Neighborhood directory that they are in.

Text List of Application Names

This example lists all applications available for hard-coded user credentials Replace *username*, *domain*, and *password* with valid strings. The URL for the ASP example is http://%webpath%/simple_1.asp.

(SIMPLE_1.ASP) Code

```
<HTML>
<HEAD>
        <TITLE>NFuse Classic Example</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<%
// Set credentials to represent the Credentials Java object, then initialize credentials
// with username, domain and password.
Set credentials = Server.CreateObject("com.citrix.nfuse.Credentials")
credentials.initialize "username", "domain", "password"

// Set gateway to represent the Wire Gateway Java object, then initialize the gateway
// using the credentials object.
Set gateway = Server.CreateObject("com.citrix.nfuse.CitrixWireGateway")
gateway.initialize credentials

// Set appEnumerator to establish getAppEnumerator object from the gateway object.
Set appEnumerator = gateway.getAppEnumerator()

// Iterate through the hasMoreAppsFlat object of appEnumerator using a While loop.
// When no more applications are available in the flat App object, the loop will terminate.
While appEnumerator.hasMoreAppsFlat("")
        // Set currentApp to the next available application.
        Set currentApp = appEnumerator.nextAppFlat("")
        // Write HTML to show the name of the current application, using the
        // getFriendlyName() property of the appEnumerator object.
        Response.Write currentApp.getFriendlyName()
        // Write an HTML break to force a new line before writing the next app (if any):
        Response.Write "<BR>"
Wend
%>
        </BODY>
</HTML>
```

Text List of Names, Details and Icons

This example lists all applications available for hard-coded user credentials, their URL-encoded names, application details, and icon locations. In addition, it embeds the icon itself into the page. Each application is delimited using the HTML horizontal rule tag (<HR>). Replace *username*, *domain*, and *password* with valid strings. The URL for the ASP example is http://%webpath%/simple_2.asp.

(SIMPLE_2.ASP) Code

```
<HTML>
<HEAD>
        <TITLE>NFuse Classic Example</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<%
// Set credentials to represent the Credentials Java object, then initialize credentials
// with username, domain and password.
Set credentials = Server.CreateObject("com.citrix.nfuse.Credentials")
credentials.initialize "username", "domain", "password"

// Set gateway to represent the Wire Gateway Java object, then initialize the gateway using
// the credentials object.
Set gateway = Server.CreateObject("com.citrix.nfuse.CitrixWireGateway")
gateway.initialize credentials

// Set appEnumerator to establish getAppEnumerator object from the gateway object.
Set appEnumerator = gateway.getAppEnumerator()

// Iterate through the hasMoreAppsFlat object of appEnumerator using a While loop.
// However, we're now adding in several other properties beyond just the getFriendlyName property.
While appEnumerator.hasMoreAppsFlat("")
        Set currentApp = appEnumerator.nextAppFlat("")
        // Write HTML to show the friendly name of the current application as before,
        // using the getFriendlyName() property of the appEnumerator object.
        Response.Write "Friendly Name: " & currentApp.getFriendlyName() & "<BR>"
        // Write out HTML to show the URL-encoded name.
        Response.Write "URL-Encoded Name: " & currentApp.getFriendlyNameUrlEncoded() & "<BR>"
        // Write out HTML to show the description text.
        Response.Write "Description: " & currentApp.getDescription() & "<BR>"
        // Write out HTML to show the URL location of the application's icon.
        Response.Write "Icon URL: " & currentApp.getIconFile() & "<BR>"
        // Use getIconFile to present the icon itself, using the HTML <IMG> tag.
        Response.Write "Icon: " & "<IMG SRC='" & currentApp.getIconFile() & "'>"
        // Put in an HTML horizontal line break before proceeding to the next application in the list.
        Response.Write "<HR>"
Wend
%>
        </BODY>
</HTML>
```

Tabular Listing of Applications, Details, and Icons

This example builds upon the first two examples to demonstrate how to take application data returned from appEnumerator and convert it into a dynamic HTML table. This example is more concentrated on correct display of the information than previous examples. The URL for the ASP example is http://%webpath%/simple_3.asp.

(SIMPLE_3.ASP) Code

```
<HTML>
<HEAD>
        <TITLE>NFuse Classic Example</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<%
// Set credentials to represent the Credentials Java object, then initialize
// credentials with username, domain and password.
Set credentials = Server.CreateObject("com.citrix.nfuse.Credentials")
credentials.initialize "user", "domain", "password"

// Set gateway to represent the Wire Gateway Java object, then initialize the
// gateway using the credentials object.
Set gateway = Server.CreateObject("com.citrix.nfuse.CitrixWireGateway")
gateway.initialize credentials

// Set appEnumerator to establish getAppEnumerator object from the gateway object.
Set appEnumerator = gateway.getAppEnumerator()

// Call appEnumerator to determine total amount of available applications for this user.
numApps = appEnumerator.getNumAppsFlat("")
// numCols will determine how many columns to use in the app listing table.
numCols = 3
%>

<TABLE BORDER=0>
<%
// Iterate through the applications until they are all found.
For i = 1 to (numApps)
       Set currentApp = appEnumerator.nextAppFlat("")
// if we have done enough columns, we start a new HTML row.
       if 1 = i Mod numCols then
             Response.Write vbCrLf & "<TR>"
       end if

// display the applications icon and friendly name.
      Response.Write "<TD>"
      Response.Write "<img src=" & currentApp.getIconFile() & "' ALIGN='absmiddle'>"
      Response.Write currentApp.getFriendlyName()
        Response.Write "</TD>" & vbCrLf

 // if we have done enough columns, we start a new HTML row.
       if (0 = i Mod numCols) Or (i = numApps) then
```

```
            Response.Write "</TR>" & vbCrLf
        end if
next
%>
</TABLE>
        </BODY>
</HTML>
```

Folder Iteration

The examples in this section expand those in the previous section; they show all of the published applications available to the user. These are listed in their respective Program Neighborhood directories, not the application directory structure built using the Citrix Management Console.

Text Listing of Root Folders

This example demonstrates simple iteration through available NFuse Classic folders off of the root level, printing out the name of each folder. The URL is http://%webpath%/folder_1.asp.

(FOLDER_1.ASP) Code

```
<HTML>
<HEAD>
        <TITLE>NFuse Classic Example</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<%
// Set credentials to represent the Credentials Java object, then initialize
// credentials with username, domain and password.
Set credentials = Server.CreateObject("com.citrix.nfuse.Credentials")
credentials.initialize "username", "domain", "password"

// Set gateway to represent the Wire Gateway Java object, then initialize the
// gateway using the credentials object.
Set gateway = Server.CreateObject("com.citrix.nfuse.CitrixWireGateway")
gateway.initialize credentials

// Set appEnumerator to establish getAppEnumerator object from the gateway object.
Set appEnumerator = gateway.getAppEnumerator()

// Iterate through the current folders using a While loop, ending when all the subfolders have been found.
while appEnumerator.hasMoreFolders("\")
      nextFolderInRoot = appEnumerator.nextFolder("\")
      Response.Write nextFolderInRoot & "<BR>"
Wend
%>
        </BODY>
</HTML>
```

Clickable Folder Navigation

This example demonstrates how to iterate simply through available NFuse Classic folders and to provide the name of each available folder as an HTML link. Navigation is provided for both subfolder selection, and for moving up from a subfolder towards the root level. The URL is http://%webpath%/folder_2.asp.

(FOLDER_2.ASP) Code

```
<HTML>
<HEAD>
<TITLE>NFuse Classic Example</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<%

// First call the App object. This allows access to methods needed later (unUrlEncode).
Set app = Server.CreateObject("com.citrix.nfuse.App")

// Set credentials to represent the Credentials Java object, then initialize
// credentials with username, domain and password.
Set credentials = Server.CreateObject("com.citrix.nfuse.Credentials")
credentials.initialize "username", "domain", "password"

// Set gateway to represent the Wire Gateway Java object, then initialize the
// gateway using the credentials object.
Set gateway = Server.CreateObject("com.citrix.nfuse.CitrixWireGateway")
gateway.initialize credentials

// Set appEnumerator to establish getAppEnumerator object from the gateway object.
Set appEnumerator = gateway.getAppEnumerator()

// This pulls the requested current folder from the user's QueryString.
currentFolder = Request.QueryString("currentFolder")

// The current folder needs to be un-URL encoded. We rely on a property of the App object
// to decode this properly.
currentFolder = app.unUrlEncode(currentFolder)

// See if anything is stored in currentFolder. If not, assume that current folder is the root (null).
if Len(currentFolder) = 0 then
        currentFolder = ""
else
        // If there is a current folder (not at root), we need to determine the parent folder.
        // This is done by parsing the currentFolder variable, taking everything prior to the last backslash.
        folderUp = Mid(currentFolder,1,InStrRev(currentFolder,Chr(92))-1)

        // The parent folder is then referenced by a link marked UP, to allow up-one-level navigation.
        // If clicked, it will refresh the page, using the parent folder as the current folder via
        // QueryString submission.
        Response.Write "<A HREF='folder_2.asp?currentFolder=" & folderUp & "'>UP</A>"
        Response.Write "<HR>"
end if
```

```
// Iterate through the current folders using a While loop, ending when all the subfolders have been found.
While appEnumerator.hasMoreFolders(currentFolder)
        nextFolderInSelected = appEnumerator.nextFolder(currentFolder)

        // The folder clicked on will refresh the page, using the selected subfolder as
        // the new current folder using a QueryString submission.
        Response.Write "<A HREF='folder_2.asp?currentFolder=" & currentFolder & "\"
        Response.Write nextFolderInSelected & ">" & nextFolderInSelected & "</A>"
        Response.Write "<BR>"
Wend
%>
</BODY>
</HTML>
```

Clickable Folder Navigation, and Application Names

This example combines the previous examples into a complete application listing page, providing both folder navigation along with text listings of available applications in the currently selected folder. The URL is http://%webpath%/folder_3.asp.

(FOLDER_3.ASP) Code

```
<HTML>
<HEAD>
<TITLE>NFuse Classic Example</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<%

// First call the App object; this allows access to methods needed later (unUrlEncode).
Set app = Server.CreateObject("com.citrix.nfuse.App")

// Set credentials to represent the Credentials Java object, then initialize
// credentials with username, domain and password.
Set credentials = Server.CreateObject("com.citrix.nfuse.Credentials")
credentials.initialize "username", "domain", "password"

// Set gateway to represent the Wire Gateway Java object, then initialize the gateway
// using the credentials object.
Set gateway = Server.CreateObject("com.citrix.nfuse.CitrixWireGateway")
gateway.initialize credentials

// Set appEnumerator to establish getAppEnumerator object from the gateway object.
Set appEnumerator = gateway.getAppEnumerator()

// Pull the requested current folder from the user's QueryString.
currentFolder = Request.QueryString("currentFolder")

// The current folder needs to be un-URL encoded. We rely on a property of the App object
// to decode this properly.
currentFolder = app.unUrlEncode(currentFolder)

// See if anything is stored in currentFolder. If not, assume the current folder is the root (null).
```

```
if Len(currentFolder) = 0 then
        currentFolder = ""
else
        // If there is a current folder (not at root), determine the parent folder. This is done
        // by parsing the currentFolder variable, taking everything prior to the last backslash.
        folderUp = Mid(currentFolder,1,InStrRev(currentFolder,Chr(92))-1)

        // The parent folder is then referenced by a link marked UP, to allow up-one-level navigation.
        // If clicked, it will refresh the page, using the parent folder as the current folder
        //using a QueryString submission.
        Response.Write "<A HREF='folder_3.asp?currentFolder=" & folderUp & ">UP</A>"
        Response.Write "<HR>"
end if

// Iterate through the current folders using a While loop, ending when all the subfolders have been found.
While appEnumerator.hasMoreFolders(currentFolder)
        nextFolderInSelected = appEnumerator.nextFolder(currentFolder)

        // The folder clicked on will refresh the page, using the selected subfolder as the new
        // current folder using a QueryString submission.
        Response.Write "<A HREF='folder_3.asp?currentFolder=" & currentFolder & "\"
        Response.Write nextFolderInSelected & ">" & nextFolderInSelected & "</A>"
        Response.Write "<BR>"
Wend
Response.Write "<HR>"

// This While loop gives a text listing of the names of all applications in the current folder.
While appEnumerator.hasMoreApps(currentFolder)
        Set currentApp = appEnumerator.nextApp(currentFolder)
        Response.Write currentApp.getFriendlyName & "<BR>"
Wend
%>
</BODY>
</HTML>
```

Launching Applications

Launchable Application Lists

This example shows how to parse a ICA template file using the TemplateParser objects. It consists of three documents: the ICA file (template_1.ica), the actual parsing file (launch_1.asp), and an ASP page that iterates through the appEnumeration object and assigns hotlinks to each application in the collection. The URL is http://%webpath%/applaunch_1.asp.

(APPLAUNCH_1.ASP) Code

```
<HTML>
<HEAD>
<TITLE>NFuse Classic Example</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<%
```

```
// First call the App object; this allows access to methods needed later (unUrlEncode).
Set app = Server.CreateObject("com.citrix.nfuse.App")

// Set credentials to represent the Credentials Java object, then initialize
// credentials with username, domain, and password.
Set credentials = Server.CreateObject("com.citrix.nfuse.Credentials")
credentials.initialize "user", "domain", "password"

// Set gateway to represent the Wire Gateway Java object, then initialize the gateway
// using the credentials object.
Set gateway = Server.CreateObject("com.citrix.nfuse.CitrixWireGateway")
gateway.initialize credentials

// Set appEnumerator to establish getAppEnumerator object from the gateway object.
Set appEnumerator = gateway.getAppEnumerator()

// Pull the requested current folder from the user's QueryString.
currentFolder = Request.QueryString("currentFolder")

// The current folder needs to be un-URL encoded. We rely on a property of the
// App object to decode this properly.
currentFolder = app.unUrlEncode(currentFolder)

// See if anything is stored in currentFolder. If not, assume the current folder is the root (null).
if Len(currentFolder) = 0 then
        currentFolder = ""
else
        // If there is a current folder (not at root), we need to determine the parent folder.
        // This is done by parsing the currentFolder variable, taking everything prior to the last backslash.
        folderUp = Mid(currentFolder,1,InStrRev(currentFolder,Chr(92))-1)

        // The parent folder is then referenced by a link marked UP, to allow up-one-level navigation.
        // If clicked, it will refresh the page, using the parent folder as the current folder
        //using a QueryString submission.
        Response.Write "<A HREF='applaunch_1.asp?currentFolder=" & folderUp & "'>UP</A>"
        Response.Write "<HR>"
end if

// Iterate through the current folders using a While loop, ending when all the subfolders have been found.
While appEnumerator.hasMoreFolders(currentFolder)
        nextFolderInSelected = appEnumerator.nextFolder(currentFolder)

        // The folder clicked on will refresh the page, using the selected subfolder as the
        // new current folder using a QueryString submission.
        Response.Write "<A HREF='applaunch_1.asp?currentFolder=" & currentFolder & "\"
        Response.Write nextFolderInSelected & "'>" & nextFolderInSelected & "</A>"
        Response.Write "<BR>"
Wend
Response.Write "<HR>"

// This While loop gives a text listing of the names of all applications in the current folder.
While appEnumerator.hasMoreApps(currentFolder)
        Set currentApp = appEnumerator.nextApp(currentFolder)
```

```
        // When writing the application name, we enclose it in an HTML link tag. This points to
        // launch_1.asp, with the current app name passed as a query string argument.
        // The app name, when used as part of a link, needs to be called as getNameUrlEncoded,
        // to prevent non-URL characters from invalidating the request.
        Response.Write "<A HREF='launch_1.asp?NFuse_Application=" & currentApp.getNameUrlEncoded
        Response.Write ">" & currentApp.getFriendlyName & "</A><BR>"
Wend
%>
</BODY>
</HTML>
```

(TEMPLATE_1.ICA) Code

```
[WFClient]
Version=2

[ApplicationServers]
[NFuse_AppName]=

[[NFuse_AppName]]
Address=[NFuse_IPV4Address]
InitialProgram=#[NFuse_AppName]
TransportDriver=TCP/IP
WinStationDriver=ICA 3.0

AutologonAllowed=ON
[NFuse_Ticket]
[NFuse_IcaWindow]
```

(LAUNCH_1.ASP) Code

```
<%
Response.Buffer=true
// Create the TemplateParser object, used to parse through template_1.ica and pass
// the resulting ICA file to the client.
Set parser = Server.CreateObject("com.citrix.nfuse.TemplateParser")

// As in the above examples, the user, domain, and password are embedded directly
// into the file. In normal operation, pass this credential information using some other
// method (HTTP POST, cookie storage, database storage, etc.).
parser.setCookieSessionFields("NFuse_User=username&NFuse_Password=password&NFuse_Domain=domain")

// Pull all information from the user's querystring. This info is used by the TemplateParser
// to determine what application and what settings to apply; in this example, it is only used to
// set the value of NFuse_Application. See the applaunch_1.asp file:
// it passes this info using a link to launch_1.asp?NFuse_Application=<appname>.
UrlSessionFields = Request.ServerVariables("QUERY_STRING")

// Set the session fields.
parser.setUrlSessionFields(UrlSessionFields)

// The statement below uses ASP server variables to determine the complete path of
// this ASP file (launch_1.asp).
```

```
CurrentTransPath = Request.ServerVariables("PATH_TRANSLATED")

// This removes the filename, leaving a complete path to the current directory.
// This path is then sent to the parser object's NFuse_TemplatesDir field.
// This directory path is used to locate the template ICA file.
parser.setSingleSessionField "NFuse_TemplatesDir",Left(CurrentTransPath,InStrRev(CurrentTransPath,"\"))

// The name of the template ICA file is then put into the NFuse_Template field.
parser.setSingleSessionField "NFuse_Template", "template_1.ica"

// if parser.Parse() returns FALSE, we know that an error has occurred, and present it to the user.
If parser.Parse() = False Then
        Response.Write "There was an error:<br><i>" & parser.getLastError() & "</i>"
Else
        // Otherwise, set the ContentType to application/x-ica (the MIME type for the Citrix ICA Client).
        Response.ContentType = "application/x-ica"
        Continue = True
        // Set the response to expire in the past, preventing any accidental caching by the web server or client.
        Response.ExpiresAbsolute = #May 1,1998 11:00:00#
        // Until the parser returns EOF (a blank line), pass the current data block to the client.

While (Continue)
        HtmlString = parser.getNextDataBlock()
        If HtmlString = "" Then
                Continue = False
        Else
                Response.Write(HtmlString)
        End If
Wend
End If
%>
```

Seamless Windows

This example demonstrates how to launch seamless windowed applications by parsing an ICA template file using the TemplateParser objects. It consists of three documents: the ICA file (template_2.ica), the actual parsing file (launch_2.asp), and an ASP page that iterates through the appEnumeration object and assigns hotlinks to each application in the collection. The URL is http://%webpath%/applaunch_2.asp.

(APPLAUNCH_2.ASP) Code

```
<HTML>
<HEAD>
<TITLE>NFuse Classic Example</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<%
// First call the App object; this allows access to methods needed later (unUrlEncode).
Set app = Server.CreateObject("com.citrix.nfuse.App")

// Set credentials to represent the Credentials Java object, then initialize
// credentials with username, domain and password.
```

```
Set credentials = Server.CreateObject("com.citrix.nfuse.Credentials")
credentials.initialize "user", "domain", "password"

// Set gateway to represent the Wire Gateway Java object, then initialize the gateway
// using the credentials object.
Set gateway = Server.CreateObject("com.citrix.nfuse.CitrixWireGateway")
gateway.initialize credentials

// Set appEnumerator to establish getAppEnumerator object from the gateway object.
Set appEnumerator = gateway.getAppEnumerator()

// Pull the requested current folder from the users's QueryString.
currentFolder = Request.QueryString("currentFolder")

// The current folder needs to be un-URL encoded. We rely on a property of the
// App object to decode this properly.
currentFolder = app.unUrlEncode(currentFolder)

// See if anything is stored in currentFolder. If not, assume the current folder is the root (null).
if Len(currentFolder) = 0 then
        currentFolder = ""
else
        // If there is a current folder (not at root), determine the parent folder.
        // This is done by parsing the currentFolder variable, taking everything prior to the last backslash.
        folderUp = Mid(currentFolder,1,InStrRev(currentFolder,Chr(92))-1)

        // The parent folder is then referenced by a link marked UP, to allow up-one-level navigation.
        // If clicked, it will refresh the page, using the parent folder as the current folder
        // using a QueryString submission.
        Response.Write "<A HREF='applaunch_2.asp?currentFolder=" & folderUp & "'>UP</A>"
        Response.Write "<HR>"
end if

// Iterate through the current folders using a While loop, ending when all the subfolders have been found.
While appEnumerator.hasMoreFolders(currentFolder)
        nextFolderInSelected = appEnumerator.nextFolder(currentFolder)

        // The folder clicked on will refresh the page, using the selected subfolder as the
        // new current folder using QueryString submission.
        Response.Write "<A HREF='applaunch_2.asp?currentFolder=" & currentFolder & "\"
        Response.Write nextFolderInSelected & "'>" & nextFolderInSelected & "</A>"
        Response.Write "<BR>"
Wend
Response.Write "<HR>"

// This while-loop gives a text listing of the names of all applications in the current folder.
While appEnumerator.hasMoreApps(currentFolder)
        Set currentApp = appEnumerator.nextApp(currentFolder)
        Response.Write "<A HREF='launch_2.asp?NFuse_Application=" & currentApp.getNameUrlEncoded
        Response.Write "'>" & currentApp.getFriendlyName & "</A><BR>"
Wend
%>
</BODY>
</HTML>
```

(TEMPLATE_2.ICA) Code

```
[WFClient]
Version=2

[ApplicationServers]
[NFuse_AppName]=

[[NFuse_AppName]]
Address=[NFuse_IPV4Address]
InitialProgram=#[NFuse_AppName]
TransportDriver=TCP/IP
WinStationDriver=ICA 3.0

AutologonAllowed=ON
[NFuse_Ticket]
[NFuse_IcaWindow]
```

(LAUNCH_2.ASP) Code

```
<%
Response.Buffer=true
// Create the TemplateParser object. It is used to parse through template_2.ica and pass the
// resulting ICA file to the client.
Set parser = Server.CreateObject("com.citrix.nfuse.TemplateParser")
// As in the above examples, the user, domain, and password are embedded directly into the file.
// In normal operation, you would pass this credential information using another method
// (HTTP POST, cookie storage, database storage, etc.).

parser.setCookieSessionFields("NFuse_User=username&NFuse_Password=password&NFuse_Domain=domain")

// Pull all information from the user's querystring. This info is used by the TemplateParser
// to determine what application and what settings to apply; in this example, it is only used
// to set the value of NFuse_Application. See the applaunch_2.asp file:
// it passes this info using a link to launch_2.asp?NFuse_Application=<appname>.
UrlSessionFields = Request.ServerVariables("QUERY_STRING")

// Set the session fields.
parser.setUrlSessionFields(UrlSessionFields)

// The statement below uses ASP server variables to determine the complete path of
// this ASP file (launch_2.asp).
CurrentTransPath = Request.ServerVariables("PATH_TRANSLATED")

// The filename is removed, leaving a complete path to the current directory.
// This path is then sent to the parser object's NFuse_TemplatesDir field.
// It is used to locate the template ICA file.
parser.setSingleSessionField "NFuse_TemplatesDir",Left(CurrentTransPath,InStrRev(CurrentTransPath,"\"))

// The name of the template ICA file is then put into the NFuse_Template field.
parser.setSingleSessionField "NFuse_Template", "template_2.ica"

// To indicate that seamless windows should be used, set the NFuse_WindowType method
// in the parser object. This will cause the [NFuse_IcaWindow] token in template_2.ica
```

```
// to be replaced with proper seamless ICA settings.
parser.setSingleSessionField "NFuse_WindowType", "seamless"

// if parser.Parse() returns FALSE, an error has occurred, and it is present it to the user.
If parser.Parse() = False Then
        Response.Write "There was an error:<br><i>" & parser.getLastError() & "</i>"
Else
        // Otherwise, set the ContentType to application/x-ica (the MIME type for the Citrix ICA Client).
        Response.ContentType = "application/x-ica"
        Continue = True

        // Set the response to expire in the past, preventing any accidental caching by the web server or client.
        Response.ExpiresAbsolute = #May 1,1998 11:00:00#

        // Until the parser returns EOF (a blank line), pass the current data block to the client.
        While (Continue)
                HtmlString = parser.getNextDataBlock()
        If HtmlString = "" Then
                Continue = False
        Else
                Response.Write(HtmlString)
        End If
        Wend
End If
%>
```

Embedding Applications

This example shows how to embed applications in a Web page by parsing an ICA template file using the TemplateParser objects. It consists of four documents: the ICA file (template_3.ica), the actual parsing file (launch_3.asp), a page to embed the application in (embed_3*x*.asp), and an ASP page that iterates through the appEnumeration object and assigns hotlinks to each application in the collection. The URL is http://%webpath%/appembed_3.asp.

There are four embed files included in this example: **Embed_3a.asp**, **Embed_3b.asp**, **Embed_3c.asp**, and **Embed_3d.asp.** Each can be called by uncommenting the corresponding lines in the file **Appembed_3.asp**. These examples each demonstrate a different method of embedding an application:

- **Embed_3a.asp**: Clicking the published application launches it in a new browser window using the Win32 ActiveX Client.
- **Embed_3b.asp**: Clicking the published application launches it in a new browser window using the Java Applet. The applet does not need to be pre-installed on the client; installation will be executed automatically, if necessary.
- **Embed_3c.asp**: Clicking the published application launches a secure, encrypted connection in a new browser window using the Java Applet. The applet does not need to be pre-installed on the client; installation will be executed automatically, if necessary.

- **Embed_3d.asp**: Clicking the published application launches the application as a plug-in in a Netscape browser.

(APPEMBED_3.ASP) Code

```
<HTML>
<HEAD>
<TITLE>NFuse Classic Example</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<%
// Set credentials to represent the Credentials Java object, then initialize
// credentials with username, domain and password.
Set credentials = Server.CreateObject("com.citrix.nfuse.Credentials")
credentials.initialize "user", "domain", "password"

// Set gateway to represent the Wire Gateway Java object, then initialize the
// gateway using the credentials object.
Set gateway = Server.CreateObject("com.citrix.nfuse.CitrixWireGateway")
gateway.initialize credentials

// Set appEnumerator to establish getAppEnumerator object from the gateway object.
Set appEnumerator = gateway.getAppEnumerator()

// Pull the requested current folder from the user's QueryString.
currentFolder = Request.QueryString("currentFolder")

// See if anything is stored in currentFolder. If not, assume the current folder is the root (null).
if Len(currentFolder) = 0 then
        currentFolder = ""
else
        // If there is a current folder (not at root), we need to determine the parent folder.
        // This is done by parsing the currentFolder variable, taking everything prior to the last backslash.

        folderUp = Mid(currentFolder,1,InStrRev(currentFolder,Chr(92))-1)
        // The parent folder is then referenced by a link marked UP, to allow up-one-level navigation.
        // If clicked, it will refresh the page, using the parent folder as the current folder
        // using a QueryString submission.
        Response.Write "<A HREF='appembed_3.asp?currentFolder=" & folderUp & "'>UP</A>"
        Response.Write "<HR>"
end if

// Iterate through the current folders using a While loop, ending when all the subfolders have been found.
While appEnumerator.hasMoreFolders(currentFolder)
        nextFolderInSelected = appEnumerator.nextFolder(currentFolder)

        // The folder clicked on will refresh the page, using the selected subfolder as the
        // new current folder using a QueryString submission.
        Response.Write "<A HREF='appembed_3.asp?currentFolder=" & currentFolder & "\"
        Response.Write nextFolderInSelected & "'>" & nextFolderInSelected & "</A>"
        Response.Write "<BR>"
Wend
Response.Write "<HR>"
```

```
// This While loop generates a text listing of the names of all applications in the current folder.
While appEnumerator.hasMoreApps(currentFolder)
        Set currentApp = appEnumerator.nextApp(currentFolder)
        //Modify to select the embedding type embed_3a=Win 32 Client  embed_3b=Java Client
        // embed_3c=Java client with encryption or embed_3d=Netscape Plugin
        // a)
        //Response.Write "<A HREF='embed_3a.asp?NFuse_Application="
        //Response.Write currentApp.getNameUrlEncoded
        //Response.Write "'" & " target=_blank" & ">" & currentApp.getFriendlyName& "</A><BR>"

        // b)
        //Response.Write "<A HREF='embed_3b.asp?NFuse_Application="
        //Response.Write currentApp.getNameUrlEncoded
        //Response.Write "'" & " target=_blank" & ">" & currentApp.getFriendlyName& "</A><BR>"

        // c)
        //Response.Write "<A HREF='embed_3c.asp?NFuse_Application="
        //Response.Write currentApp.getNameUrlEncoded
        //Response.Write "'" & " target=_blank" & ">" & currentApp.getFriendlyName& "</A><BR>"

        // d)
        //Response.Write "<A HREF='embed_3d.asp?NFuse_Application="
        //Response.Write currentApp.getNameUrlEncoded
        //Response.Write "'" & " target=_blank" & ">" & currentApp.getFriendlyName& "</A><BR>"
Wend
%>
</BODY>
</HTML>
```

(EMBED_3a.ASP) Code

```
<OBJECT classid="clsid:238f6f83-b8b4-11cf-8771-00a024541ee3"
        WIDTH=800 HEIGHT=600
        // Insert the path to the Client database on your webserver.
        CODEBASE="http://yourwebserver/your directory/icaweb/en/ica32.exe#Version=6,30,1000">
        <param name="ICAFile"
        value="launch_3.asp?NFuse_Application=<%=Request.QueryString("NFuse_Application")%>">
        <param name="Start" value='Auto'>
        <param name="Border" value='Off'>
</OBJECT>
```

(EMBED_3b.ASP) Code

```
<!-- Insert the path to the Client database on your webserver.-->
<APPLET
        codebase="http://yourwebserver/your directory/icaweb/en/icajava"
        code='com.citrix.JICA'
        WIDTH=800 HEIGHT=600
        archive='JICAEngN.jar'>
        <param name=cabinets value='JICAEngM.cab'>
        <param name=Start value='Auto'>
        <param name=icafile
```

```
        value="launch_3.asp?NFuse_Application=<%=Request.QueryString("NFuse_Application")%>">
</APPLET>
```

(EMBED_3c.ASP) Code

```
<!-- Insert the path to the Client database on your webserver.-->
<APPLET
        codebase="http://yourwebserver/your directory/icaweb/en/icajava"
        code='com.citrix.JICA'
        WIDTH=800 HEIGHT=600
        ARCHIVE='JICAEngN.jar,cryptoJN.jar,JICA-sicaN.jar'>
        <param name=cabinets value='JICAEngM.cab,cryptojM.cab,JICA-sicaM.cab'>
        <param name=Start value='Auto'>
        <param name=icafile
        value="launch_3.asp?NFuse_Application=<%=Request.QueryString("NFuse_Application")%>">
        <param name=encryptionlevel value='5'>
        <param name=usedefaultencryption value='off'>
</APPLET>
```

(EMBED_3d.ASP) Code

(EMBED_3d.JSP) Code

```
        <EMBED
        SRC="launch_3.asp?NFuse_Application=<%=Request.QueryString("NFuse_Application")%>"
        // Insert the path to the Netscape plugin page on your webserver.
        PLUGINSPAGE='http://yourwebserver/yourpluginpage.htm'
        TYPE='application/x-ica'
        WIDTH=800 HEIGHT=600
        START='Auto'
        BORDER='0'>
</EMBED>
```

(TEMPLATE_3.ICA) Code

```
[WFClient]
Version=2

[ApplicationServers]
[NFuse_AppName]=

[[NFuse_AppName]]
Address=[NFuse_IPV4Address]
InitialProgram=#[NFuse_AppName]
TransportDriver=TCP/IP
WinStationDriver=ICA 3.0

AutologonAllowed=ON
[NFuse_Ticket]
[NFuse_IcaWindow]
```

(LAUNCH_3.ASP) Code

```
<%
Response.Buffer=true
// Create the TemplateParser object. This is used to parse through template_3.ica and
```

```
// pass the resulting ICA file to the client.
Set parser = Server.CreateObject("com.citrix.nfuse.TemplateParser")

// As in the above examples, the user, domain, and password are embedded directly into the file.
// In normal operation, you would pass this credential information using another method, such as
// (HTTP POST, cookie storage, database storage, etc.).
parser.setCookieSessionFields("NFuse_User=username&NFuse_Password=password&NFuse_Domain=domain")

// Pull all information from the user's querystring. This information is used by the TemplateParser
// to determine what application and what settings to apply; in this example, it is only used to set
// the value of NFuse_Application. See the appembed_3.asp file:
// it passes this info using a link to launch_3.asp?NFuse_Application=<appname>.
UrlSessionFields = Request.ServerVariables("QUERY_STRING")

// Set the session fields below.
parser.setUrlSessionFields(UrlSessionFields)

// The statement below uses ASP server variables to figure out the complete path of
// this ASP file (launch_3.asp).
CurrentTransPath = Request.ServerVariables("PATH_TRANSLATED")

// The file name is removed, leaving a complete path to the current directory.
// This path is then sent to the parser object's NFuse_TemplatesDir field.
// It will use this directory path to locate the template ICA file.
parser.setSingleSessionField "NFuse_TemplatesDir",Left(CurrentTransPath,InStrRev(CurrentTransPath,"\"))

// The name of the template ICA file is then put into the NFuse_Template field.
parser.setSingleSessionField "NFuse_Template", "template_3.ica"

// if parser.Parse() returns FALSE, an error has occurred. This is presented to the user.
If parser.Parse() = False Then
        Response.Write "There was an error:<br><i>" & parser.getLastError() & "</i>"
Else
        // Otherwise, set the ContentType to application/x-ica (the MIME type for the Citrix ICA Client).
        Response.ContentType = "application/x-ica"
        Continue = True

        // Set the response to expire in the past, preventing any accidental caching by the web server or client.
        Response.ExpiresAbsolute = #May 1,1998 11:00:00#
        // Until the parser returns EOF (a blank line), pass the current data block to the client.
While (Continue)
        HtmlString = parser.getNextDataBlock()
        If HtmlString = "" Then
                Continue = False
        Else
                Response.Write(HtmlString)
        End If
Wend
End If
%>
```

Disabling Ticketing

This example shows how to disable Ticketing. As in the previous examples, it parses an ICA template file using the TemplateParser objects. It consists of three documents: the ICA file (template_4.ica), the actual parsing file (launch_4.asp), and an ASP page that iterates through the appEnumeration object and assigns hotlinks to each application in the collection. The URL is http://%webpath%/applaunch_4.asp.

(APPLAUNCH_4.ASP) Code

```
<HTML>
<HEAD>
<TITLE>NFuse Classic Example</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<%
// First call the App object; this allows access to methods needed later (unUrlEncode).
Set app = Server.CreateObject("com.citrix.nfuse.App")

// Set credentials to represent the Credentials Java object, then initialize
// credentials with username, domain, and password.
Set credentials = Server.CreateObject("com.citrix.nfuse.Credentials")
credentials.initialize "user", "domain", "password"

// Set gateway to represent the Wire Gateway Java object, then initialize the
// gateway using the credentials object.
Set gateway = Server.CreateObject("com.citrix.nfuse.CitrixWireGateway")
gateway.initialize credentials

// Set appEnumerator to establish getAppEnumerator object from the gateway object.
Set appEnumerator = gateway.getAppEnumerator()

// Pull the requested current folder from the user's QueryString.
currentFolder = Request.QueryString("currentFolder")

// The current folder needs to be un-URL encoded. We rely on a property of the App object
// to decode this properly.
currentFolder = app.unUrlEncode(currentFolder)

// See if anything is stored in currentFolder. If not, assume the current folder is the root (null).
if Len(currentFolder) = 0 then
        currentFolder = ""
else
        // If there is a current folder (not at root), determine the parent folder.
        // This is done by parsing the currentFolder variable, taking everything prior to the last backslash.
        folderUp = Mid(currentFolder,1,InStrRev(currentFolder,Chr(92))-1)
        // The parent folder is then referenced by a link marked UP, to allow up-one-level navigation.
        // If clicked, it will refresh the page, using the parent folder as the current folder
        // using a QueryString submission.
        Response.Write "<A HREF='applaunch_4.asp?currentFolder=" & folderUp & "'>UP</A>"
        Response.Write "<HR>"
end if
```

```
// Iterate through the current folders using a While loop, ending when all the subfolders have been found.
While appEnumerator.hasMoreFolders(currentFolder)
        nextFolderInSelected = appEnumerator.nextFolder(currentFolder)
        // The folder clicked on will refresh the page, using the selected subfolder as the
        // new current folder using a QueryString submission.
        Response.Write "<A HREF='applaunch_4.asp?currentFolder=" & currentFolder & "\"
        Response.Write nextFolderInSelected & ">" & nextFolderInSelected & "</A>"
        Response.Write "<BR>"
Wend
Response.Write "<HR>"

// This While loop gives a text listing of the names of all applications in the current folder.
While appEnumerator.hasMoreApps(currentFolder)
        Set currentApp = appEnumerator.nextApp(currentFolder)
        Response.Write "<A HREF='launch_4.asp?NFuse_Application=" & currentApp.getNameUrlEncoded
        Response.Write ">" & currentApp.getFriendlyName & "</A><BR>"
Wend
%>
</BODY>
</HTML>
```

(TEMPLATE_4.ICA) Code

```
[WFClient]
Version=2

[ApplicationServers]
[NFuse_AppName]=

[[NFuse_AppName]]
Address=[NFuse_IPV4Address]
InitialProgram=#[NFuse_AppName]
TransportDriver=TCP/IP
WinStationDriver=ICA 3.0

AutologonAllowed=ON
;Replace the NFuse_Ticket token with the following. This will, if the application's encryption level is
; "basic", use plain-text credentials in the ICA file (username and domain in clear text, password scrambled)
; instead of ticketing.
<[NFuse_IFSESSIONFIELD sessionfield="NFUSE_ENCRYPTIONLEVEL"value="basic"]>
Username=[NFuse_User]
Domain=[NFuse_Domain]
Password=[NFuse_PasswordScrambled]
<[/NFuse_IFSESSIONFIELD]>

[NFuse_IcaWindow]
```

(LAUNCH_4.ASP) Code

```
<%
Response.Buffer=true
// Create the TemplateParser object, used to parse through template_4.ica and pass the
// resulting ICA file to the client.
Set parser = Server.CreateObject("com.citrix.nfuse.TemplateParser")
```

```
// As in the above examples, the user, domain, and password are embedded directly into the file.
// In normal operation, you would pass this credential information using another method
// (HTTP POST, cookie storage, database storage, etc.).
parser.setCookieSessionFields("NFuse_User=username&NFuse_Password=password&NFuse_Domain=domain")

// Pull all information from the user's querystring. This info is used by the TemplateParser
// to determine what application and what settings to apply; in this example, it is only used
// to set the value of NFuse_Application. See the applaunch_4.asp file:
// it passes this info using a link to launch_4.asp?NFuse_Application=<appname>.
UrlSessionFields = Request.ServerVariables("QUERY_STRING")

// Set the session fields below.
parser.setUrlSessionFields(UrlSessionFields)

// The statement below uses ASP server variables to figure out the complete path of this
// ASP file (launch_4.asp).
CurrentTransPath = Request.ServerVariables("PATH_TRANSLATED")

// The filename is removed, leaving a complete path to the current directory.
// This path is then sent to the parser object's NFuse_TemplatesDir field.
// It will use this directory path to locate the template ICA file.
parser.setSingleSessionField "NFuse_TemplatesDir",Left(CurrentTransPath,InStrRev(CurrentTransPath,"\"))

// The name of the template ICA file is then put into the NFuse_Template field.
parser.setSingleSessionField "NFuse_Template", "template_4.ica"

// if parser.Parse() returns FALSE, an error has occurred. It is presented it to the user.
If parser.Parse() = False Then
        Response.Write "There was an error:<br><i>" & parser.getLastError() & "</i>"
Else
        // Otherwise, set the ContentType to application/x-ica (the MIME type for the Citrix ICA Client).
        Response.ContentType = "application/x-ica"
        Continue = True

        // Set the response to expire in the past, preventing any accidental caching by the web server or client.
        Response.ExpiresAbsolute = #May 1,1998 11:00:00#

        // Until the parser returns EOF (a blank line), pass the current data block to the client.
        While (Continue)
                HtmlString = parser.getNextDataBlock()
                If HtmlString = "" Then
                        Continue = False
                Else
                        Response.Write(HtmlString)
                End If
        Wend
End If
%>
```

Passing User Credentials/Login

This example shows how to provide login capabilities using a standard HTML form to get the information and store it in cookies. This information is used to

parse a ICA template file using the TemplateParser objects. It consists of four documents: the ICA file (template_5.ica), the actual parsing file (launch_5.asp), and an ASP page that iterates through the appEnumeration object and assigns hotlinks to each application in the collection (applaunch_5). The URL is http://%webpath%/login_5.htm. The URL for the ASP example is http://%webpath%/applaunch_5.asp.

(LOGIN_5.HTM) Code

```
<HTML>
<HEAD>
<TITLE>NFuse Classic Example</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<!-- This is a simple HTML form that submits a username, domain and password directly to applaunch_5.asp using
the HTTP POST method.-->
<form method="POST" action="applaunch_5.asp" name="NFuseForm">
User:
<input type="text" name="user">
Domain:
<input type="text" name="domain">
Password:
<input type="password" name="password">
<input type=submit>
</form>
</body>
</html>
```

(APPLAUNCH_5.ASP) Code

```
<HTML>
<HEAD>
<TITLE>NFuse Classic Example</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<%
// If we have a username posted from the login form, we set the credentials cookie.
if len (Request.Form("user")) >0 then
        // Store the login information in a cookie.
        NFuseCookie="NFuse_User=" + Request.Form("user")
        NFuseCookie = NFuseCookie + "&NFuse_Domain=" + Request.Form("domain")
        NFuseCookie = NFuseCookie + "&NFuse_Password=" + Request.Form("password")
        Response.Cookies("NFuseData") = NFuseCookie
end if

//get login information
user = Request.Cookies("NFuseData")("NFuse_User")
domain = Request.Cookies("NFuseData")("NFuse_Domain")
password = Request.Cookies("NFuseData")("NFuse_Password")

// First call the App object; this allows access to methods needed later (unUrlEncode).
Set app = Server.CreateObject("com.citrix.nfuse.App")

// Set credentials to represent the Credentials Java object, then initialize
```

```
// credentials with username, domain and password from the login form.
Set credentials = Server.CreateObject("com.citrix.nfuse.Credentials")
credentials.initialize user,domain,password

// Set gateway to represent the Wire Gateway Java object, then initialize the
// gateway using the credentials object.
Set gateway = Server.CreateObject("com.citrix.nfuse.CitrixWireGateway")
gateway.initialize credentials

// Set appEnumerator to establish getAppEnumerator object from the gateway object.
Set appEnumerator = gateway.getAppEnumerator()

// Pull the requested current folder from the user's QueryString.
currentFolder = Request.QueryString("currentFolder")

// The current folder needs to be un-URL encoded. We rely on a property of the
// App object to decode this properly.
currentFolder = app.unUrlEncode(currentFolder)

// See if anything is stored in currentFolder. If not, assume the current folder is the root (null).
if Len(currentFolder) = 0 then
        currentFolder = ""
else
        // If there is a current folder (not at root), determine the parent folder.
        // This is done by parsing the currentFolder variable, taking everything prior to the last backslash.
        folderUp = Mid(currentFolder,1,InStrRev(currentFolder,Chr(92))-1)
        // The parent folder is then referenced by a link marked UP, to allow up-one-level navigation.
        // If clicked, it will refresh the page, using the parent folder as the current folder
        //using a QueryString submission.
        Response.Write "<A HREF='applaunch_5.asp?currentFolder=" & folderUp & "'>UP</A>"
        Response.Write "<HR>"
end if

// Iterate through the current folders using a While loop, ending when all the subfolders have been found.
While appEnumerator.hasMoreFolders(currentFolder)
        nextFolderInSelected = appEnumerator.nextFolder(currentFolder)
        // The folder clicked on will refresh the page, using the selected subfolder as
        // the new current folder using a QueryString submission.
        Response.Write "<A HREF='applaunch_5.asp?currentFolder=" & currentFolder & "\"
        Response.Write nextFolderInSelected & "'>" & nextFolderInSelected & "</A>"
        Response.Write "<BR>"
Wend
Response.Write "<HR>"

// This While loop gives a text listing of the names of all applications in the current folder.
While appEnumerator.hasMoreApps(currentFolder)
        Set currentApp = appEnumerator.nextApp(currentFolder)
        Response.Write "<A HREF='launch_5.asp?NFuse_Application=" & currentApp.getNameUrlEncoded
        Response.Write "'>" & currentApp.getFriendlyName & "</A><BR>"
Wend
%>
</BODY>
</HTML>
```

(TEMPLATE_5.ICA) Code

```
[WFClient]
Version=2

[ApplicationServers]
[NFuse_AppName]=

[[NFuse_AppName]]
Address=[NFuse_IPV4Address]
InitialProgram=#[NFuse_AppName]
TransportDriver=TCP/IP
WinStationDriver=ICA 3.0

AutologonAllowed=ON
[NFuse_Ticket]
[NFuse_IcaWindow]
```

(LAUNCH_5.ASP) Code

```
<%
Response.Buffer=true
// We create the TemplateParser object, used to parse through template_5.ica and pass
// the resulting ICA file to the client.
Set parser = Server.CreateObject("com.citrix.nfuse.TemplateParser")

// The user, domain, and password are all read from the cookie created in applaunch_5.
CookStr = Request.Cookies("NFuseData")
parser.setCookieSessionFields(CookStr)

// Pull all information from the user's querystring. This info is used by the TemplateParser
// to determine what application and what settings to apply; in this example, it is only used
// to set the value of NFuse_Application. See the applaunch_5.asp file:
// it passes this info using a link to launch_5.asp?NFuse_Application=<appname>.
UrlSessionFields = Request.ServerVariables("QUERY_STRING")

// We set the session fields below.
parser.setUrlSessionFields(UrlSessionFields)

// The statement below uses ASP server variables to figure out the complete path of
// this ASP file (launch_5.asp).
CurrentTransPath = Request.ServerVariables("PATH_TRANSLATED")

// The filename is removed, leaving a complete path to the current directory.
// This is then sent to the parser object's NFuse_TemplatesDir field.
// It will use this directory path to locate the template ICA file.
parser.setSingleSessionField "NFuse_TemplatesDir",Left(CurrentTransPath,InStrRev(CurrentTransPath,"\"))

// The name of the template ICA file is then put into the NFuse_Template field.
parser.setSingleSessionField "NFuse_Template", "template_5.ica"

// if parser.Parse() returns FALSE, an error has occurred, and it is presented to the user.
If parser.Parse() = False Then
        Response.Write "There was an error:<br><i>" & parser.getLastError() & "</i>"
```

```
Else
    // Otherwise, we set the ContentType to application/x-ica (the MIME type for the Citrix ICA Client).
    Response.ContentType = "application/x-ica"
    Continue = True
    // Set the response to expire in the past, preventing any accidental caching by the web server or client.
    Response.ExpiresAbsolute = #May 1,1998 11:00:00#
    // Until the parser returns EOF (a blank line), pass the current data block to the client.
    While (Continue)
        HtmlString = parser.getNextDataBlock()
        If HtmlString = "" Then
            Continue = False
        Else
            Response.Write(HtmlString)
        End If
    Wend
End If
%>
```

Caching

This example demonstrates how caching works. Applications are cached and used to parse a ICA template file using the TemplateParser objects. It consists of three documents: the ICA file (template_6.ica), the actual parsing file (launch_6.asp), and an ASP page that iterates through the appEnumeration object and assigns hotlinks to each application in the collection. The URL is http://%webpath%/applaunch_6.asp.

(APPLAUNCH_6.ASP) Code

```
<HTML>
<HEAD>
<TITLE>NFuse Classic Example</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<%
// First call the App object; this allows access to methods needed later (unUrlEncode).
Set app = Server.CreateObject("com.citrix.nfuse.App")

// Set credentials to represent the Credentials Java object, then initialize
// credentials with username, domain and password.

Set credentials = Server.CreateObject("com.citrix.nfuse.Credentials")
credentials.initialize "username", "domain", "password"

// Set gateway to represent the Wire Gateway Java object, then initialize the
// gateway using the credentials object.
Set gateway = Server.CreateObject("com.citrix.nfuse.CitrixWireGateway")
gateway.initialize credentials

// Use the global application object to store the cache; lock it first to ensure no one else is using it.
Application.Lock

// if we do not have a cache we create one.
If IsObject(Application("CitrixNfuseCache")) = false Then
```

```
        Set citrixCache = Server.CreateObject("com.citrix.nfuse.AppListCache")
        Set Application("CitrixNfuseCache") = citrixCache
        // If we already have a cache, we reference it.
Else
        Set citrixCache = Application("CitrixNfuseCache")
End If
Application.UnLock

// Create a name for this object stored in the cache.
key = app.unUrlEncode(domain) & "\" & app.unUrlEncode(user) & "\" & app.unUrlEncode(password)
key = UCase(key)

// Try to get the data from the cache first.
If citrixCache.contain(key) = true Then
        Set apps = citrixCache.retrieveFromCache(key)
End If
On Error Resume Next

// If the data is not in the cache or it is expired, we get it from the wire.
If IsEmpty(apps) = true Then
        Set apps = gateway.getAppDataList()
        citrixCache.addToCache key, apps
ElseIf apps.isExpired() = true Then
        Set apps = gateway.getAppDataList()
        citrixCache.addToCache key, apps
End If

// Set appEnumerator to establish getAppEnumerator object.
Set appEnumerator = apps.getAppEnumerator()

// Pull the requested current folder from the user's QueryString.
currentFolder = Request.QueryString("currentFolder")

// The current folder needs to be un-URL encoded. We rely on a property of the App object
// to decode this properly.
currentFolder = app.unUrlEncode(currentFolder)

// See if anything is stored in currentFolder. If not, assume the current folder is the root (null).
if Len(currentFolder) = 0 then
        currentFolder = ""
else
        // If there is a current folder (not at root), we need to determine the parent folder.
        // This is done by parsing the currentFolder variable, taking everything prior to the last backslash.
        folderUp = Mid(currentFolder,1,InStrRev(currentFolder,Chr(92))-1)

        // The parent folder is then referenced by a link marked UP, to allow up-one-level navigation.
        // If clicked, it will refresh the page, using the parent folder as the current folder
        //using a QueryString submission.
        Response.Write "<A HREF='applaunch_6.asp?currentFolder=" & folderUp & "'>UP</A>"
        Response.Write "<HR>"
end if

// Iterate through the current folders using a While loop, ending when all the subfolders have been found.
While appEnumerator.hasMoreFolders(currentFolder)
```

```
        nextFolderInSelected = appEnumerator.nextFolder(currentFolder)

        // The folder clicked on will refresh the page, using the selected subfolder as the
        // new current folder using a QueryString submission.
        Response.Write "<A HREF='applaunch_6.asp?currentFolder=" & currentFolder & "\"
        Response.Write nextFolderInSelected & ">" & nextFolderInSelected & "</A>"
        Response.Write "<BR>"
Wend
Response.Write "<HR>"

// This While loop will give a text listing of the names of all applications in the current folder.
While appEnumerator.hasMoreApps(currentFolder)
        Set currentApp = appEnumerator.nextApp(currentFolder)
        Response.Write "<A HREF='launch_6.asp?NFuse_Application=" & currentApp.getNameUrlEncoded
        Response.Write ">" & currentApp.getFriendlyName & "</A><BR>"
Wend
%>
</BODY>
</HTML>
```

(TEMPLATE_6.ICA) Code

```
[WFClient]
Version=2

[ApplicationServers]
[NFuse_AppName]=

[[NFuse_AppName]]
Address=[NFuse_IPV4Address]
InitialProgram=#[NFuse_AppName]
TransportDriver=TCP/IP
WinStationDriver=ICA 3.0

AutologonAllowed=ON
[NFuse_Ticket]
[NFuse_IcaWindow]
```

(LAUNCH_6.ASP) Code

```
<%
Response.Buffer=true
// Create the TemplateParser object; this is used to parse through template_6.ica and
// pass the resulting ICA file to the client.

Set parser = Server.CreateObject("com.citrix.nfuse.TemplateParser")
// As in the above examples, the user, domain, and password are embedded directly into the file.
// In normal operation, you would pass this credential information using a some other method
// (HTTP POST, cookie storage, database storage, etc.).
parser.setCookieSessionFields("NFuse_User=username&NFuse_Password=password&NFuse_Domain=domain")

// Pull all information from the user's querystring. This info is used by the TemplateParser
// to determine what application and what settings to apply; in this example, it is only used
// to set the value of NFuse_Application. See the applaunch_6.asp file:
// it passes this info using a link to launch_6.asp?NFuse_Application=<appname>.
```

```
UrlSessionFields = Request.ServerVariables("QUERY_STRING")

// Set the session fields below.
parser.setUrlSessionFields(UrlSessionFields)

// The statement below uses ASP server variables to figure out the complete path
// of this ASP file (launch_6.asp).
CurrentTransPath = Request.ServerVariables("PATH_TRANSLATED")

// The filename is removed, leaving a complete path to the current directory.
// This is then sent to the parser object's NFuse_TemplatesDir field.
// It will use this directory path to locate the template ICA file.
parser.setSingleSessionField "NFuse_TemplatesDir",Left(CurrentTransPath,InStrRev(CurrentTransPath,"\"))

// The name of the template ICA file is then put into the NFuse_Template field.
parser.setSingleSessionField "NFuse_Template", "template_6.ica"

// To indicate that seamless windows should be used, just set the NFuse_WindowType method
// in the parser object. This will cause the [NFuse_IcaWindow] token in template_6.ica
// to be replaced with proper seamless ICA settings.
parser.setSingleSessionField "NFuse_WindowType", "seamless"

// if parser.Parse() returns FALSE, an error has occurred, and it is presented it to the user.
If parser.Parse() = False Then
        Response.Write "There was an error:<br><i>" & parser.getLastError() & "</i>"
Else
        // Otherwise, set the ContentType to application/x-ica (the MIME type for the Citrix ICA Client).
        Response.ContentType = "application/x-ica"
        Continue = True

        // Set the response to expire in the past, preventing any accidental caching by the web server or client.
        Response.ExpiresAbsolute = #May 1,1998 11:00:00#

        // Until the parser returns EOF (a blank line), pass the current data block to the client.
        While (Continue)
                HtmlString = parser.getNextDataBlock()
                If HtmlString = "" Then
                        Continue = False
                Else
                        Response.Write(HtmlString)
                End If
        Wend
End If
%>
```

JSP Examples

Simple (Flat) Application Iteration

The examples in this section show all published applications available to the user in a flat structure, regardless of the Program Neighborhood directory that they are in.

Text List of Application Names

This example lists all applications available for hard-coded user credentials Replace *username*, *domain*, and *password* with valid strings. The URL for this example is http://%webpath%/simple_1.jsp.

(SIMPLE_1.JSP) Code

```
<%@ page import = "com.citrix.nfuse.*" %>


<%
try
{
%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<HTML>
<HEAD>
        <TITLE>NFuse Classic Example</TITLE>
</HEAD>
<BODY BGCOLOR='#FFFFFF'>
<%
// Declare local variables.
Credentials credentials = null;
CitrixWireGateway gateway = null;
AppEnumerator appEnumerator = null;
App currentApp = null;

// Set credentials to represent the Credentials Java object, then initialize
// credentials with username, domain, and password.
credentials = new Credentials();
credentials.initialize ("username", "domain", "password");

// Set gateway to represent the Wire Gateway Java object, then initialize the
// gateway using the credentials object.
gateway = new CitrixWireGateway();
gateway.initialize(credentials);

// Set appEnumerator to establish getAppEnumerator object from the gateway object.
appEnumerator = gateway.getAppEnumerator();

// Iterate through the hasMoreAppsFlat object of appEnumerator using a While loop.
```

```
// When no more applications are available in the flat App object, the loop will terminate.
while (appEnumerator.hasMoreAppsFlat("")) {
        // Set currentApp to the next available application.
        currentApp = appEnumerator.nextAppFlat("");
        // Write out HTML to show the name of the current application, using the
        // getFriendlyName() property of the appEnumerator object.
        out.println (currentApp.getFriendlyName());
        // Write out an HTML break to force a newline before writing the next application (if any).
        out.println ("<BR>");
}
%>
        </BODY>
</HTML>


<%
}
catch (Exception e)
{
        System.out.println("catch ERROR");
}
%>
```

Text List of Names, Details and Icons

This example lists all applications available for hard-coded user credentials, their URL-encoded names, application details, and icon locations. In addition, it embeds the icon itself into the page. Each application is delimited using the HTML horizontal rule tag (<HR>). Replace *username*, *domain*, and *password* with valid strings. The URL for this example is http://%webpath%/simple_2.jsp.

(SIMPLE_2.JSP) Code

```
<%@ page import = "com.citrix.nfuse.*" %>


<%
try
{
%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<HTML>
<HEAD>
        <TITLE>NFuse Classic Example</TITLE>
</HEAD>
<BODY BGCOLOR='#FFFFFF'>
<%
//Declare local variables.
Credentials credentials = null;
```

```
CitrixWireGateway gateway = null;
AppEnumerator appEnumerator = null;
App currentApp = null;

// Set credentials to represent the Credentials Java object, then initialize
// credentials with username, domain, and password.
credentials = new Credentials();
credentials.initialize ("username", "domain", "password");

// Set gateway to represent the Wire Gateway Java object, then initialize the
// gateway using the credentials object.
gateway = new CitrixWireGateway();
gateway.initialize(credentials);

// Set appEnumerator to establish getAppEnumerator object from the gateway object.
appEnumerator = gateway.getAppEnumerator();

// Iterate through the hasMoreAppsFlat object of appEnumerator using a While loop.
// However, we're now adding in several other properties beyond just the getFriendlyName property.
while (appEnumerator.hasMoreAppsFlat("")) {
        currentApp = appEnumerator.nextAppFlat("");
        // Write out HTML to show the friendly  name of the current application, using the
        // getFriendlyName() property of the appEnumerator object.
        out.println ("Friendly Name: " + currentApp.getFriendlyName() + "<BR>");
        // Write out HTML to show the URL-encoded name.
        out.println ("URL-Encoded Name: " + currentApp.getFriendlyNameUrlEncoded() + "<BR>");
        // Write HTML to show the description text.
        out.println ("Description: " + currentApp.getDescription() + "<BR>");
        // Write out HTML to show the URL  location of the application's icon.
        out.println ("Icon URL: " + currentApp.getIconFile() + "<BR>");
        // Use getIconFile to present the icon itself, using the HTML <IMG> tag.
        out.println ("Icon:  " + "<IMG SRC='" + currentApp.getIconFile() + "'>");
        // Lastly, put in an HTML horizontal line break before proceeding to the next application in the list.
        out.println ("<HR>");
}
%>
        </BODY>
</HTML>


<%
}
catch (Exception e)
{
        System.out.println("catch ERROR");
}
%>
```

Tabular Listing of Applications, Details, and Icons

This example builds upon the first two examples to demonstrate how to take application data returned from appEnumerator and convert it into a dynamic HTML table. This example is more concentrated on correct display of the

information than previous examples. The URL for this example is http://%webpath%/simple_3.jsp.

(SIMPLE_3.JSP) Code

```
<%@ page import = "com.citrix.nfuse.*" %>


<%
try
{
%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<HTML>
<HEAD>
        <TITLE>NFuse Classic Example</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<%
// Declare local variables.
Credentials credentials = null;
CitrixWireGateway gateway = null;
AppEnumerator appEnumerator = null;
App currentApp = null;
int numApps, numCols;

// Set credentials to represent the Credentials Java object, then initialize
// credentials with username, domain, and password.
credentials = new Credentials();
credentials.initialize ("username", "domain", "password");

// Set gateway to represent the Wire Gateway Java object, then initialize
// the gateway using the credentials object.
gateway = new CitrixWireGateway();
gateway.initialize(credentials);

// Set appEnumerator to establish getAppEnumerator object from the gateway object.
appEnumerator = gateway.getAppEnumerator();

// Call appEnumerator to determine total amount of available applications for this user.
numApps = appEnumerator.getNumAppsFlat("");

// numCols will determine how many columns to use in the app listing table. Change this
// to desired columns, or determine progmatically.
numCols = 3;
%>

<TABLE BORDER=0>
<%

// Iterate through the applications until all are found.
for (int i = 1; i<numApps+1; i++)
```

```
{
        currentApp = appEnumerator.nextAppFlat("");
        // If we have done enough columns, we start a new HTML row.
        if (1 == i%numCols )
        {
                out.println ('\r' + "<TR>");
        }

        // Display the applications icon and friendly name.
        out.println ("<TD>");
        out.println ("<img src=" + currentApp.getIconFile() + "' ALIGN='absmiddle'>");
        out.println ( currentApp.getFriendlyName() );
        out.println ("</TD>" + '\r');

        // If we have done enough columns, we start a new HTML row.
        if ((0 == i%numCols) || (i == numApps))
        {
            out.println ("</TR>" + '\r');
        }
}
%>
</TABLE>
        </BODY>
</HTML>

<%
}
catch (Exception e)
{
        System.out.println("catch ERROR");
}
%>
```

Folder Iteration

The examples in this section expand those in the previous section; they show all of the published applications available to the user. These are listed in their respective Program Neighborhood directories, not the application directory structure built using the Citrix Management Console.

Text Listing of Root Folders

This example demonstrates simple iteration through available NFuse Classic folders off of the root level, printing out the name of each folder. The URL for this example is http://%webpath%/folder_1.jsp.

(FOLDER_1.JSP) Code

```
<%@ page import = "com.citrix.nfuse.*" %>

<%
try
```

```
{
%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<HTML>
<HEAD>
        <TITLE>NFuse Classic Example</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">

<%

// Declare local variables.
Credentials credentials = null;
CitrixWireGateway gateway = null;
AppEnumerator appEnumerator = null;
String nextFolderInRoot;

// Set credentials to represent the Credentials Java object, then initialize
// credentials with username, domain, and password.
credentials = new Credentials();
credentials.initialize ("username", "domain", "password");

// Set gateway to represent the Wire Gateway Java object, then initialize the gateway
// using the credentials object.
gateway = new CitrixWireGateway();
gateway.initialize(credentials);

// Set appEnumerator to establish getAppEnumerator object from the gateway object.
appEnumerator = gateway.getAppEnumerator();

// Iterate through the available root folders using a While loop, ending when all subfolders have been found.
while ( appEnumerator.hasMoreFolders("\\"))
{
      nextFolderInRoot = appEnumerator.nextFolder("\\");
      out.println (nextFolderInRoot + "<BR>");
}
%>
        </BODY>
</HTML>




<%
}
catch (Exception e)
{
        System.out.println("catch ERROR");
}
%>
```

Clickable Folder Navigation

This example demonstrates how to iterate simply through available NFuse Classic folders and to provide the name of each available folder as an HTML link. Navigation is provided for both subfolder selection, and for moving up from a subfolder towards the root level. The URL for this example is http://%webpath%/folder_2.jsp.

(FOLDER_2.JSP) Code

```
<%@ page import = "com.citrix.nfuse.*" %>

<%
try
{
%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<HTML>
<HEAD>
        <TITLE>NFuse Classic Example</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<%
// Declare local variables.
Credentials credentials = null;
CitrixWireGateway gateway = null;
AppEnumerator appEnumerator = null;
App app = null;
String currentFolder,folderUp,nextFolderInSelected;

// First call the App object; this allows access to methods needed later.
app = new App();

// Set credentials to represent the Credentials Java object, then initialize
// credentials with username, domain, and password.
credentials = new Credentials();
credentials.initialize ("username", "domain", "password");

// Set gateway to represent the Wire Gateway Java object, then initialize the gateway
// using the credentials object.
gateway = new CitrixWireGateway();
gateway.initialize(credentials);

// Set appEnumerator to establish getAppEnumerator object from the gateway object.
appEnumerator = gateway.getAppEnumerator();

// Pull the requested current folder from the user's QueryString.
currentFolder = request.getParameter("currentFolder");

// The current folder needs to be un-URL encoded. We rely on a property of the App object to
// decode this properly.
```

```
currentFolder = app.unUrlEncode(currentFolder);

// See if anything is stored in currentFolder. If not, assume the current folder is the root (null).
if (currentFolder == null || currentFolder.length() == 0)
{
        currentFolder = "";
}
else
{
        // If there is a current folder (not at root), determine the parent folder. This is
        // done by parsing the currentFolder variable, taking everything prior to the last backslash.
      folderUp = currentFolder.substring(0,currentFolder.lastIndexOf("\\"));

        // The parent folder is then referenced by a link marked UP, to allow up-one-level navigation.
        // If clicked, it will refresh the page, using the parent folder as the current folder
        // using a QueryString submission:
        out.println ("<A HREF='folder_2.jsp?currentFolder=" + folderUp + "'>UP</A>");
        out.println ("<HR>");
}

// Iterate through the current folders using a While loop, ending when all subfolders have been found.
while (appEnumerator.hasMoreFolders(currentFolder))
{
        nextFolderInSelected = appEnumerator.nextFolder(currentFolder);
        // The folder clicked on will refresh the page, using the selected subfolder as the new
        // current folder using a QueryString submission.
        out.println ("<A HREF='folder_2.jsp?currentFolder=" + currentFolder + "\\" );
        out.println (nextFolderInSelected + "'>" +nextFolderInSelected + "</A>");
        out.println ( "<BR>");
}
%>
</BODY>
</HTML>


<%
}
catch (Exception e)
{
        System.out.println("catch ERROR");
}
%>
```

Clickable Folder Navigation, and Application Names

This example combines the previous examples into a complete application listing page, providing both folder navigation along with text listings of available applications in the currently selected folder. The URL for this example is http://%webpath%/folder_3.jsp.

(FOLDER_3.JSP) Code

```
<%@ page import = "com.citrix.nfuse.*" %>
```

```
<%
try
{
%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<HTML>
<HEAD>
        <TITLE>NFuse Classic Example</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<%
// Declare local variables.
Credentials credentials = null;
CitrixWireGateway gateway = null;
AppEnumerator appEnumerator = null;
App currentApp,app = null;
String currentFolder,folderUp,nextFolderInSelected;

// First call the App object; this allows access to methods needed later.

app = new App();

// Set credentials to represent the Credentials Java object, then initialize
// credentials with username, domain, and password.
credentials = new Credentials();
credentials.initialize ("username", "domain", "password");

// Set gateway to represent the Wire Gateway Java object, then initialize the gateway
// using the credentials object.
gateway = new CitrixWireGateway();
gateway.initialize(credentials);

// Set appEnumerator to establish getAppEnumerator object from the gateway object.
appEnumerator = gateway.getAppEnumerator();

// Pull the requested current folder from the user's QueryString.
currentFolder = request.getParameter("currentFolder");

// The current folder needs to be un-URL encoded. We rely on a property of the App object to
// decode this properly.
currentFolder = app.unUrlEncode(currentFolder);

// See if anything is stored in currentFolder. If not, assume the current folder is the root (null).
if (currentFolder == null || currentFolder.length() == 0)
{
        currentFolder = "";
}
else
{
        // If there is a current folder (not at root), determine the parent folder. This is done by parsing
```

```
        // the currentFolder variable, taking everything prior to the last backslash.
      folderUp = currentFolder.substring(0,currentFolder.lastIndexOf("\\"));

        // The parent folder is then referenced by a link marked UP, to allow up-one-level navigation.
        // If clicked, it will refresh the page, using the parent folder as the current folder
        // using a QueryString submission:
      out.println ("<A HREF='folder_3.jsp?currentFolder=" + folderUp + ">UP</A>");
      out.println ("<HR>");
}

// Iterate through the current folders using a While loop, ending when all subfolders have been found.
while (appEnumerator.hasMoreFolders(currentFolder))
{
        nextFolderInSelected = appEnumerator.nextFolder(currentFolder);
        // The folder clicked on will refresh the page, using the selected subfolder as the new
        // current folder using a QueryString submission.
        out.println ("<A HREF='folder_3.jsp?currentFolder=" + currentFolder + "\\");
        out.println (nextFolderInSelected + ">" +nextFolderInSelected + "</A>");
        out.println ( "<BR>");
}
out.println("<HR>");

// This While loop gives a text listing of the names of all applications in the current folder.
while (appEnumerator.hasMoreApps(currentFolder))
{
        currentApp = appEnumerator.nextApp(currentFolder);
        out.println (currentApp.getFriendlyName() + "<BR>");
}
%>
</BODY>
</HTML>


<%
}
catch (Exception e)
{
        System.out.println("catch ERROR");
}
%>
```

Launching Applications

Launchable Application Lists

This example shows how to parse a ICA template file using the TemplateParser objects. It consists of three documents: the ICA file (template_1.ica), the actual parsing file (launch_1.jsp), and a JSP page that iterates through the appEnumeration object and assigns hotlinks to each application in the collection. The URL for this example is http://%webpath%/applaunch_1.jsp.

Note If you are running Jserv 1.2.x or earlier, change the line in each launch_x.jsp file from

CurrentTransPath = config.getServletContext().getRealPath(request.getServletPath());

to

CurrentTransPath = request.getRealPath(request.getRequestURI());

(APPLAUNCH_1.JSP) Code

```
<%@ page import = "com.citrix.nfuse.*" %>

<%
try
{
%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<HTML>
<HEAD>
        <TITLE>NFuse Classic Example</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<%
//Declare local variables.
Credentials credentials = null;
CitrixWireGateway gateway = null;
AppEnumerator appEnumerator = null;
App currentApp,app = null;
String currentFolder,folderUp,nextFolderInSelected;

// First call the App object; this allows access to methods needed later (unUrlEncode).
app = new App();

// Set credentials to represent the Credentials Java object, then initialize
// credentials with username, domain, and password.
credentials = new Credentials();
credentials.initialize ("username", "domain", "password");

// Set gateway to represent the Wire Gateway Java object, then initialize the gateway
// using the credentials object.
gateway = new CitrixWireGateway();
```

```
gateway.initialize(credentials);

// Set appEnumerator to establish getAppEnumerator object from the gateway object.
appEnumerator = gateway.getAppEnumerator();

// Pull the requested current folder from the user's QueryString.
currentFolder = request.getParameter("currentFolder");

// The current folder needs to be un-URL encoded. We rely on a property of the
// App object to decode this properly.
currentFolder = app.unUrlEncode(currentFolder);

// See if anything is stored in currentFolder. If not, assume the current folder is the root (null).
if (currentFolder == null || currentFolder.length() == 0)
{
        currentFolder = "";
}
else
{
        // If there is a current folder (not at root), determine the parent folder.
        // This is done by parsing the currentFolder variable, taking everything prior to the last backslash.
        folderUp = currentFolder.substring(0,currentFolder.lastIndexOf("\\"));

        // The parent folder is then referenced by a link marked UP, to allow up-one-level navigation.
        // If clicked, it will refresh the page, using the parent folder as the current folder
        // using a QueryString submission.
        out.println ("<A HREF='applaunch_1.jsp?currentFolder=" + folderUp + "'>UP</A>");
        out.println ("<HR>");
}

// Iterate through the current folders using a While loop, ending when all the subfolders have been found.
while (appEnumerator.hasMoreFolders(currentFolder))
{
        nextFolderInSelected = appEnumerator.nextFolder(currentFolder);
        // The folder clicked on will refresh the page, using the selected subfolder as the
        // new current folder using a QueryString submission.
        out.println ("<A HREF='applaunch_1.jsp?currentFolder=" + currentFolder + "\\");
        out.println (nextFolderInSelected + "'>" +nextFolderInSelected + "</A>");
        out.println ( "<BR>");
}
out.println("<HR>");

// This While loop gives a text listing of the names of all applications in the current folder.
while (appEnumerator.hasMoreApps(currentFolder))
{
        currentApp = appEnumerator.nextApp(currentFolder);

        // When writing the application name, we enclose it in an HTML link tag. This points to
        // launch_1.jsp, with the current app name passed as a query string argument.
        // The app name, when used as part of a link, needs to be called as getNameUrlEncoded,
        // to prevent non-URL characters from invalidating the request.
        out.println ("<A HREF=launch_1.jsp?NFuse_Application=" + currentApp.getNameUrlEncoded() );
```

```
        out.println (">" + currentApp.getFriendlyName() + "</A><BR>");
}
%>
</BODY>
</HTML>


<%
}
catch (Exception e)
{
        System.out.println("catch ERROR");
}
%>
```

(TEMPLATE_1.ICA) Code

```
[WFClient]
Version=2

[ApplicationServers]
[NFuse_AppName]=

[[NFuse_AppName]]
Address=[NFuse_IPV4Address]
InitialProgram=#[NFuse_AppName]
TransportDriver=TCP/IP
WinStationDriver=ICA 3.0

AutologonAllowed=ON
[NFuse_Ticket]
[NFuse_IcaWindow]
```

(LAUNCH_1.JSP) Code

```
<%@ page import = "com.citrix.nfuse.*" %>
<%
try{
%>


<%

// Declare local variables.
TemplateParser parser;
String CookStr;
String queryString;
String HtmlString;
String UrlSessionFields;
String CurrentTransPath;
Cookie[] cookies;

// Create the TemplateParser object, used to parse through template_1.ica and pass
// the resulting ICA file to the client.
parser = new TemplateParser();
```

```
// As in the above examples, the user, domain, and password are embedded directly
// into the file. In normal operation, pass this credential information using another
// method (HTTP POST, cookie storage, database storage, etc.).
parser.setCookieSessionFields("NFuse_User=username&NFuse_Password=password&NFuse_Domain=domain");

// Pull all information from the user's querystring. This info is used by the TemplateParser
// to determine what application and what settings to apply; in this example, it is only used to
// set the value of NFuse_Application. See the applaunch_1.jsp file below:
// it passes this info using a link to launch_1.jsp?NFuse_Application=<appname>.
queryString = request.getQueryString();

// Set the session fields.
parser.setUrlSessionFields(queryString);

// The statement below uses JSP server variables to determine the complete path of
// this JSP file (launch_1.jsp).
CurrentTransPath = config.getServletContext().getRealPath(request.getServletPath());



// This removes the filename, leaving a complete path to the current directory.
// This path is then sent to the parser object's NFuse_TemplatesDir field.
// This directory path is used to locate the template ICA file.
if (CurrentTransPath.indexOf('\\') == -1)
{
        CurrentTransPath = CurrentTransPath.substring(0,CurrentTransPath.lastIndexOf('/'));
}
else
{
        CurrentTransPath = CurrentTransPath.substring(0,CurrentTransPath.lastIndexOf('\\'));
}

// The name of the template ICA file is then put into the NFuse_Template field.
parser.setSingleSessionField("NFuse_TemplatesDir",CurrentTransPath);
parser.setSingleSessionField("NFuse_Template", "template_1.ica");

// if parser.Parse() returns FALSE, we know that an error has occurred, and present it to the user.
if (!parser.Parse())
{
        out.println("There was an error:<br><i>" + parser.getLastError() + "</i>");
}
else
{
        // Otherwise, set the ContentType to application/x-ica (the MIME type for the Citrix ICA Client).
        response.setHeader("Content-Type", "application/x-ica");
        boolean Continue = true;
        HtmlString = "";
        // Until the parser returns EOF (a blank line), pass the current data block to the client.
        while (Continue)
        {
                HtmlString = parser.getNextDataBlock();
                if (HtmlString.length() == 0)
                {
```

```
                    Continue = false;
                }
                else
                {
                    out.println(HtmlString);
                }
        }
}

%>

<%
}
catch (PNException e)

{
        System.out.println("catch ERROR");
}
%>
```

Seamless Windows

This example demonstrates how to launch seamless windowed applications by parsing an ICA template file using the TemplateParser objects. It consists of three documents: the ICA file (template_2.ica), the actual parsing file (launch_2.jsp), and a JSP page that iterates through the appEnumeration object and assigns hotlinks to each application in the collection. The URL for the JSP example is http://%webpath%/applaunch_2.jsp.

(APPLAUNCH_2.JSP) Code

```
<%@ page import = "com.citrix.nfuse.*" %>

<%
try
{
%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<HTML>
<HEAD>
        <TITLE>NFuse Classic Example</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<%
// Declare local variables.
Credentials credentials = null;
CitrixWireGateway gateway = null;
AppEnumerator appEnumerator = null;
App currentApp,app = null;
String currentFolder,folderUp,nextFolderInSelected;

// Call the App object; this allows access to methods needed later (unUrlEncode).
```

```
app = new App();

// Set credentials to represent the Credentials Java object, then initialize
// credentials with username, domain, and password.
credentials = new Credentials();
credentials.initialize ("username", "domain", "password");

// Set gateway to represent the Wire Gateway Java object, then initialize the gateway
// using the credentials object.
gateway = new CitrixWireGateway();
gateway.initialize(credentials);

// Set appEnumerator to establish getAppEnumerator object from the gateway object.
appEnumerator = gateway.getAppEnumerator();

// Pull the requested current folder from the user's QueryString.
currentFolder = request.getParameter("currentFolder");

// The current folder needs to be un-URL encoded. We rely on a property of the
// App object to decode this properly.
currentFolder = app.unUrlEncode(currentFolder);

// See if anything is stored in currentFolder. If not, assume the current folder is the root (null).
if (currentFolder == null || currentFolder.length() == 0)
{
        currentFolder = "";
}
else
{
        // If there is a current folder (not at root), we need to determine the parent folder.
        // This is done by parsing the currentFolder variable, taking everything prior to the last backslash.
      folderUp = currentFolder.substring(0,currentFolder.lastIndexOf("\\"));

        // The parent folder is then referenced by a link marked UP, to allow up-one-level navigation.
        // If clicked, it will refresh the page, using the parent folder as the current folder
        // using a QueryString submission.
      out.println ("<A HREF='applaunch_2.jsp?currentFolder=" + folderUp + "'>UP</A>");
      out.println ("<HR>");
}

// Iterate through the current folders using a While loop, ending when all the subfolders have been found.
while (appEnumerator.hasMoreFolders(currentFolder))
{
        nextFolderInSelected = appEnumerator.nextFolder(currentFolder);
        // The folder clicked on will refresh the page, using the selected subfolder as the
        // new current folder using a QueryString submission.
        out.println ("<A HREF='applaunch_2.jsp?currentFolder=" + currentFolder + "\\");
        out.println (nextFolderInSelected + "'>" +nextFolderInSelected + "</A>");
        out.println ( "<BR>");
}
out.println("<HR>");

// This While loop gives a text listing of the names of all applications in the current folder.
```

```
while (appEnumerator.hasMoreApps(currentFolder))
{
        currentApp = appEnumerator.nextApp(currentFolder);

        // When writing the application name, we enclose it in an HTML link tag. This points to
        // launch_2.jsp, with the current app name passed as a query string argument.
        // The app name, when used as part of a link, needs to be called as getNameUrlEncoded,
        // to prevent non-URL characters from invalidating the request.
        out.println ("<A HREF=launch_2.jsp?NFuse_Application=" + currentApp.getNameUrlEncoded() );
        out.println (">" + currentApp.getFriendlyName() + "</A><BR>");
}
%>
</BODY>
</HTML>


<%
}
catch (Exception e)
{
        System.out.println("catch ERROR");
}
%>
```

(TEMPLATE_2.ICA) Code

```
[WFClient]
Version=2

[ApplicationServers]
[NFuse_AppName]=

[[NFuse_AppName]]
Address=[NFuse_IPV4Address]
InitialProgram=#[NFuse_AppName]
TransportDriver=TCP/IP
WinStationDriver=ICA 3.0

AutologonAllowed=ON
[NFuse_Ticket]
[NFuse_IcaWindow]
```

(LAUNCH_2.JSP) Code

```
<%@ page import = "com.citrix.nfuse.*" %>
<%
try{
%>


<%

// Declare local variables.
TemplateParser parser;
```

```
String CookStr;
String queryString;
String HtmlString;
String UrlSessionFields;
String CurrentTransPath;
Cookie[] cookies;

// Create the TemplateParser object, used to parse through template_1.ica and pass
// the resulting ICA file to the client.
parser = new TemplateParser();

// As in the above examples, the user, domain, and password are embedded directly
// into the file. In normal operation, pass this credential information using another
// method (HTTP POST, cookie storage, database storage, etc.).
parser.setCookieSessionFields("NFuse_User=username&NFuse_Password=password&NFuse_Domain=domain");

// Pull all information from the user's querystring. This info is used by the TemplateParser
// to determine what application and what settings to apply; in this example, it is only used to
// set the value of NFuse_Application. See the applaunch_2.jsp file below:
// it passes this info using a link to launch_2.jsp?NFuse_Application=<appname>.
queryString = request.getQueryString();

// Set the session fields.
parser.setUrlSessionFields(queryString);

// The statement below uses ASP server variables to determine the complete path of
// this ASP file (launch_2.jsp).
CurrentTransPath = config.getServletContext().getRealPath(request.getServletPath());


// This removes the filename, leaving a complete path to the current directory.
// This path is then sent to the parser object's NFuse_TemplatesDir field.
// This directory path is used to locate the template ICA file.
if (CurrentTransPath.indexOf('\\') == -1)
{
        CurrentTransPath = CurrentTransPath.substring(0,CurrentTransPath.lastIndexOf('/'));
}
else
{
        CurrentTransPath = CurrentTransPath.substring(0,CurrentTransPath.lastIndexOf('\\'));
}
parser.setSingleSessionField("NFuse_TemplatesDir",CurrentTransPath);

// The name of the template ICA file is then put into the NFuse_Template field.
parser.setSingleSessionField("NFuse_Template", "template_2.ica");

// To indicate that seamless windows should be used, set the NFuse_WindowType method
// in the parser object. This will cause the [NFuse_IcaWindow] token in template_2.ica
// to be replaced with proper seamless ICA settings.
parser.setSingleSessionField ("NFuse_WindowType", "seamless");

// if parser.Parse() returns FALSE, we know that an error has occurred, and present it to the user.
if (!parser.Parse())
{
```

```
        out.println("There was an error:<br><i>" + parser.getLastError() + "</i>");
}
else
{
        // Otherwise, set the ContentType to application/x-ica (the MIME type for the Citrix ICA Client).
        response.setHeader("Content-Type", "application/x-ica");
        boolean Continue = true;
        HtmlString = "";
        // Until the parser returns EOF (a blank line), pass the current data block to the client.
        while (Continue)
        {
                HtmlString = parser.getNextDataBlock();
                if (HtmlString.length() == 0)
                {
                        Continue = false;
                }
                else
                {
                        out.println(HtmlString);
                }
        }
}

%>

<%
}
catch (PNException e)

{
        System.out.println("catch ERROR");
}
%>
```

Embedding Applications

This example shows how to embed applications in a Web page by parsing an ICA template file using the TemplateParser objects. It consists of four documents: the ICA file (template_3.ica), the actual parsing file (launch_3.jsp), a page to embed the application in (embed_3*x*.jsp), and a JSP page that iterates through the appEnumeration object and assigns hotlinks to each application in the collection. The URL for this example is http://%webpath%/appembed_3.jsp.

There are four embed files included in this example: **Embed_3a.jsp**, **Embed_3b.jsp**, **Embed_3c.jsp**, and **Embed_3d.jsp**. Each can be called by uncommenting the corresponding lines in the file **Appembed_3.jsp**. These examples each demonstrate a different method of embedding an application:

- **Embed_3a.jsp**: Clicking the published application launches it in a new browser window using the Win32 ActiveX Client.

- **Embed_3b.jsp**: Clicking the published application launches it in a new browser window using the Java Applet. The applet does not need to be pre-installed on the client; installation will be executed automatically, if necessary.
- **Embed_3c.jsp**: Clicking the published application launches a secure, encrypted connection in a new browser window using the Java Applet. The applet does not need to be pre-installed on the client; installation will be executed automatically, if necessary.
- **Embed_3d.jsp**: Clicking the published application launches the application as a plug-in in a Netscape browser.

(APPEMBED_3.JSP) Code

```
<%@ page import = "com.citrix.nfuse.*" %>

<%
try
{
%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<HTML>
<HEAD>
        <TITLE>NFuse Classic Example</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<%
// Declare local variables.
Credentials credentials = null;
CitrixWireGateway gateway = null;
AppEnumerator appEnumerator = null;
App currentApp,app = null;
String currentFolder,folderUp,nextFolderInSelected;

// First call the App object; this allows access to methods needed later (unUrlEncode).
app = new App();

// Set credentials to represent the Credentials Java object, then initialize
// credentials with username, domain, and password.
credentials = new Credentials();
credentials.initialize ("username", "domain", "password");

// Set gateway to represent the Wire Gateway Java object, then initialize the gateway
// using the credentials object.
gateway = new CitrixWireGateway();
gateway.initialize(credentials);

// Set appEnumerator to establish getAppEnumerator object from the gateway object.
appEnumerator = gateway.getAppEnumerator();

// Pull the requested current folder from the user's QueryString.
currentFolder = request.getParameter("currentFolder");
```

```
// The current folder needs to be un-URL encoded. We rely on a property of the
// App object to decode this properly.
currentFolder = app.unUrlEncode(currentFolder);

// See if anything is stored in currentFolder. If not, assume the current folder is the root (null).
if (currentFolder == null || currentFolder.length() == 0)
{
        currentFolder = "";
}
else
{
        // If there is a current folder (not at root), determine the parent folder.
        // This is done by parsing the currentFolder variable, taking everything prior to the last backslash.
      folderUp = currentFolder.substring(0,currentFolder.lastIndexOf("\\"));

        // The parent folder is then referenced by a link marked UP, to allow up-one-level navigation.
        // If clicked, it will refresh the page, using the parent folder as the current folder
        // using a QueryString submission.
      out.println ("<A HREF='appembed_3.jsp?currentFolder=" + folderUp + ">UP</A>");
      out.println ("<HR>");
}

// Iterate through the current folders using a While loop, ending when all the subfolders have been found.
while (appEnumerator.hasMoreFolders(currentFolder))
{
        nextFolderInSelected = appEnumerator.nextFolder(currentFolder);
        // The folder clicked on will refresh the page, using the selected subfolder as the
        // new current folder using QueryString submission.
        out.println ("<A HREF='appembed_3.jsp?currentFolder=" + currentFolder + "\\");
        out.println (nextFolderInSelected + ">" +nextFolderInSelected + "</A>");
        out.println ( "<BR>");
}
out.println("<HR>");

// This While loop gives a text listing of the names of all applications in the current folder.
while (appEnumerator.hasMoreApps(currentFolder))
{
        currentApp = appEnumerator.nextApp(currentFolder);

        //Modify to select the embedding type embed_3a=Win 32 Client  embed_3b=Java Client
        // embed_3c=Java client with encryption, embed_3d=Netscape Plugin
        // a)
        //out.println ("<A HREF='embed_3a.jsp?NFuse_Application=");
        //out.println (currentApp.getNameUrlEncoded());
        //out.println ( "'" + " target=_blank" + ">" + currentApp.getFriendlyName() + "</A><BR>");

        // b)
        //out.println ("<A HREF='embed_3b.jsp?NFuse_Application=");
        //out.println (currentApp.getNameUrlEncoded());
        //out.println ("'" + " target=_blank" + ">" + currentApp.getFriendlyName() + "</A><BR>" );
```

```
        // c)
        //out.println ("<A HREF='embed_3c.jsp?NFuse_Application=");
        //out.println (currentApp.getNameUrlEncoded());
        //out.println ("'" + " target=_blank" + ">" + currentApp.getFriendlyName() + "</A><BR>");

        // d)
        //out.println ("<A HREF='embed_3d.jsp?NFuse_Application=");
        //out.println (currentApp.getNameUrlEncoded());
        //out.println ("'" + " target=_blank" + ">" + currentApp.getFriendlyName() + "</A><BR>");

}

%>
</BODY>
</HTML>


<%
}
catch (Exception e)
{
        System.out.println("catch ERROR");
}
%>
```

(EMBED_3a.JSP) Code

```
<OBJECT classid="clsid:238f6f83-b8b4-11cf-8771-00a024541ee3"
        WIDTH=800 HEIGHT=600
        // Insert the path to the Client database on your webserver.
        CODEBASE="http://yourwebserver/your directory/icaweb/en/ica32t.exe#Version=6,30,1000">
        <param name="ICAFile"
        value="launch_3.jsp?NFuse_Application=<%=request.getParameter("NFuse_Application")%>">
        <param name="Start" value='Auto'>
        <param name="Border" value='Off'>
</OBJECT>
```

(EMBED_3b.JSP) Code

```
<!-- Insert the path to the Client database on your webserver.-->
<APPLET
        codebase="http://yourwebserver/your directory/icaweb/en/icajava"
        code='com.citrix.JICA'
        WIDTH=800 HEIGHT=600
        archive='JICAEngN.jar'>
        <param name=cabinets value='JICAEngM.cab'>
        <param name=Start value='Auto'>
        <param name=icafile
        value="launch_3.jsp?NFuse_Application=<%=request.getParameter("NFuse_Application")%>">
</APPLET>
```

(EMBED_3c.JSP) Code

```
<!-- Insert the path to the Client database on your webserver.-->
<APPLET
      codebase="http://yourwebserver/your directory/icaweb/en/icajava"
      code='com.citrix.JICA'
      WIDTH=800 HEIGHT=600
      ARCHIVE='JICAEngN.jar,cryptoJN.jar,JICA-sicaN.jar'>
      <param name=cabinets value='JICAEngM.cab,cryptojM.cab,JICA-sicaM.cab'>
      <param name=Start value='Auto'>
      <param name=icafile
      value="launch_3.jsp?NFuse_Application=<%=request.getParameter("NFuse_Application")%>">
      <param name=encryptionlevel value='5'>
      <param name=usedefaultencryption value='off'>
</APPLET>
```

(EMBED_3d.JSP) Code

```
<EMBED SRC="launch_3.jsp?NFuse_Application=<%=request.getParameter("NFuse_Application")%>"
      // Insert the path to the Netscape plugin page on your webserver.
      PLUGINSPAGE='http://yourwebserver/yourpluginpage.htm'
      TYPE='application/x-ica'
      WIDTH=800 HEIGHT=600
      START='Auto'
      BORDER='0'>
</EMBED>
```

(TEMPLATE_3.ICA) Code

```
[WFClient]
Version=2

[ApplicationServers]
[NFuse_AppName]=

[[NFuse_AppName]]
Address=[NFuse_IPV4Address]
InitialProgram=#[NFuse_AppName]
TransportDriver=TCP/IP
WinStationDriver=ICA 3.0

AutologonAllowed=ON
[NFuse_Ticket]
[NFuse_IcaWindow]
```

(LAUNCH_3.JSP) Code

```
<%@ page import = "com.citrix.nfuse.*" %>
<%
try{
%>
```

```
<%

// Declare local variables.
TemplateParser parser;
String CookStr;
String queryString;
String HtmlString;
String UrlSessionFields;
String CurrentTransPath;
Cookie[] cookies;

// Create the TemplateParser object, used to parse through template_1.ica and pass
// the resulting ICA file to the client.
parser = new TemplateParser();

// As in the above examples, the user, domain, and password are embedded directly
// into the file. In normal operation, pass this credential information using another
// method (HTTP POST, cookie storage, database storage, etc.).
parser.setCookieSessionFields("NFuse_User=username&NFuse_Password=password&NFuse_Domain=domain");

// Pull all information from the user's querystring. This info is used by the TemplateParser
// to determine what application and what settings to apply; in this example, it is only used to
// set the value of NFuse_Application. See the applaunch_3.jsp file below:
// it passes this info using a link to launch_3.jsp?NFuse_Application=<appname>.
queryString = request.getQueryString();

// Set the session fields.
parser.setUrlSessionFields(queryString);

// The statement below uses JSP server variables to determine the complete path of
// this JSP file (launch_3.jsp).
CurrentTransPath = config.getServletContext().getRealPath(request.getServletPath());


// This removes the filename, leaving a complete path to the current directory.
// This path is then sent to the parser object's NFuse_TemplatesDir field.
// This directory path is used to locate the template ICA file.
if (CurrentTransPath.indexOf('\\') == -1)
{
        CurrentTransPath = CurrentTransPath.substring(0,CurrentTransPath.lastIndexOf('/'));
}
else
{
        CurrentTransPath = CurrentTransPath.substring(0,CurrentTransPath.lastIndexOf('\\'));
}
parser.setSingleSessionField("NFuse_TemplatesDir",CurrentTransPath);

// The name of the template ICA file is then put into the NFuse_Template field.
parser.setSingleSessionField("NFuse_Template", "template_3.ica");

// if parser.Parse() returns FALSE, we know that an error has occurred, and present it to the user.
if (!parser.Parse())
{
        out.println("There was an error:<br><i>" + parser.getLastError() + "</i>");
```

```
}
else
{
        // Otherwise, set the ContentType to application/x-ica (the MIME type for the Citrix ICA Client).
        response.setHeader("Content-Type", "application/x-ica");
        boolean Continue = true;
        HtmlString = "";
        // Until the parser returns EOF (a blank line), pass the current data block to the client.
        while (Continue)
        {
                HtmlString = parser.getNextDataBlock();
                if (HtmlString.length() == 0)
                {
                        Continue = false;
                }
                else
                {
                        out.println(HtmlString);
                }
        }
}

%>

<%
}
catch (PNException e)

{
        System.out.println("catch ERROR");
}
%>
```

Disabling Ticketing

This example shows how to disable ticketing by parsing an ICA template file using the TemplateParser objects. It consists of three documents: the ICA file (template_4.ica), the actual parsing file (launch_4.jsp), and a JSP page that iterates through the appEnumeration object and assigns hotlinks to each application in the collection. The URL for this example is http://%webpath%/applaunch_4.jsp.

(APPLAUNCH_4.JSP) Code

```
<%@ page import = "com.citrix.nfuse.*" %>

<%
try
{
%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
```

```
<HTML>
<HEAD>
        <TITLE>NFuse Classic Example</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<%
// Declare local variables.
Credentials credentials = null;
CitrixWireGateway gateway = null;
AppEnumerator appEnumerator = null;
App currentApp,app = null;
String currentFolder,folderUp,nextFolderInSelected;

// First call the App object; this allows access to methods needed later (unUrlEncode).
app = new App();

// Set credentials to represent the Credentials Java object, then initialize
// credentials with username, domain, and password.
credentials = new Credentials();
credentials.initialize ("username", "domain", "password");

// Set gateway to represent the Wire Gateway Java object, then initialize the gateway
// using the credentials object.
gateway = new CitrixWireGateway();
gateway.initialize(credentials);

// Set appEnumerator to establish getAppEnumerator object from the gateway object.
appEnumerator = gateway.getAppEnumerator();

// Pull the requested current folder from the user's QueryString.
currentFolder = request.getParameter("currentFolder");

// The current folder needs to be un-URL encoded. We rely on a property of the
// App object to decode this properly.
currentFolder = app.unUrlEncode(currentFolder);

// See if anything is stored in currentFolder. If not, assume the current folder is the root (null).
if (currentFolder == null || currentFolder.length() == 0)
{
        currentFolder = "";
}
else
{
        // If there is a current folder (not at root), we need to determine the parent folder.
        // This is done by parsing the currentFolder variable, taking everything prior to the last backslash.
        folderUp = currentFolder.substring(0,currentFolder.lastIndexOf("\\"));

        // The parent folder is then referenced by a link marked UP, to allow up-one-level navigation.
        // If clicked, it will refresh the page, using the parent folder as the current folder
        // using a QueryString submission.
        out.println ("<A HREF='applaunch_4.jsp?currentFolder=" + folderUp + "'>UP</A>");
        out.println ("<HR>");
}
```

```
// Iterate through the current folders using a While loop, ending when all the subfolders have been found.
while (appEnumerator.hasMoreFolders(currentFolder))
{
        nextFolderInSelected = appEnumerator.nextFolder(currentFolder);
        // The folder clicked on will refresh the page, using the selected subfolder as the
        // new current folder using a QueryString submission.
        out.println ("<A HREF='applaunch_4.jsp?currentFolder=" + currentFolder + "\\");
        out.println (nextFolderInSelected + ">" +nextFolderInSelected + "</A>");
        out.println ( "<BR>");
}
out.println("<HR>");

// This While loop gives a text listing of the names of all applications in the current folder.
while (appEnumerator.hasMoreApps(currentFolder))
{
        currentApp = appEnumerator.nextApp(currentFolder);

        // When writing the application name, we enclose it in an HTML link tag. This points to
        // launch_4.jsp, with the current app name passed as a query string argument.
        // The app name, when used as part of a link, needs to be called as getNameUrlEncoded,
        // to prevent non-URL characters from invalidating the request.
        out.println ("<A HREF=launch_4.jsp?NFuse_Application=" + currentApp.getNameUrlEncoded() );
        out.println (">" + currentApp.getFriendlyName() + "</A><BR>");
}
%>
</BODY>
</HTML>


<%
}
catch (Exception e)
{
        System.out.println("catch ERROR");
}
%>
```

(TEMPLATE_4.ICA) Code

```
[WFClient]
Version=2

[ApplicationServers]
[NFuse_AppName]=

[[NFuse_AppName]]
Address=[NFuse_IPV4Address]
InitialProgram=#[NFuse_AppName]
TransportDriver=TCP/IP
WinStationDriver=ICA 3.0

AutologonAllowed=ON
```

```
; Replace the NFuse_Ticket token with the following. This will, if the application's encryption level is
; "basic", use plain-text credentials in the ICA file (username and domain in clear text, password scrambled)
; instead of ticketing.
<[NFuse_IFSESSIONFIELD sessionfield="NFUSE_ENCRYPTIONLEVEL"value="basic"]>
Username=[NFuse_User]
Domain=[NFuse_Domain]
Password=[NFuse_PasswordScrambled]
<[/NFuse_IFSESSIONFIELD]>

[NFuse_IcaWindow]
```

(LAUNCH_4.JSP) Code

```
<%@ page import = "com.citrix.nfuse.*" %>
<%
try{
%>


<%

// Declare local variables.
TemplateParser parser;
String CookStr;
String queryString;
String HtmlString;
String UrlSessionFields;
String CurrentTransPath;
Cookie[] cookies;

// Create the TemplateParser object, used to parse through template_1.ica and pass
// the resulting ICA file to the client.
parser = new TemplateParser();

// As in the above examples, the user, domain, and password are embedded directly
// into the file. In normal operation, pass this credential information using another
// method (HTTP POST, cookie storage, database storage, etc.).
parser.setCookieSessionFields("NFuse_User=username&NFuse_Password=password&NFuse_Domain=domain");

// Pull all information from the user's querystring. This info is used by the TemplateParser
// to determine what application and what settings to apply; in this example, it is only used to
// set the value of NFuse_Application. See the applaunch_4.jsp file below:
// it passes this info using a link to launch_4.jsp?NFuse_Application=<appname>.
queryString = request.getQueryString();

// Set the session fields.
parser.setUrlSessionFields(queryString);

// The statement below uses JSP server variables to determine the complete path of
// this JSP file (launch_4.jsp).
CurrentTransPath = config.getServletContext().getRealPath(request.getServletPath());
```

```
// This removes the filename, leaving a complete path to the current directory.
// This path is then sent to the parser object's NFuse_TemplatesDir field.
// This directory path is used to locate the template ICA file.
if (CurrentTransPath.indexOf('\\') == -1)
{
        CurrentTransPath = CurrentTransPath.substring(0,CurrentTransPath.lastIndexOf('/'));
}
else
{
        CurrentTransPath = CurrentTransPath.substring(0,CurrentTransPath.lastIndexOf('\\'));
}
parser.setSingleSessionField("NFuse_TemplatesDir",CurrentTransPath);

// The name of the template ICA file is then put into the NFuse_Template field.
parser.setSingleSessionField("NFuse_Template", "template_4.ica");

// if parser.Parse() returns FALSE, we know that an error has occurred, and present it to the user.
if (!parser.Parse())
{
        out.println("There was an error:<br><i>" + parser.getLastError() + "</i>");
}
else
{
        // Otherwise, set the ContentType to application/x-ica (the MIME type for the Citrix ICA Client).
        response.setHeader("Content-Type", "application/x-ica");
        boolean Continue = true;
        HtmlString = "";
        // Until the parser returns EOF (a blank line), pass the current data block to the client.
        while (Continue)
        {
                HtmlString = parser.getNextDataBlock();
                if (HtmlString.length() == 0)
                {
                        Continue = false;
                }
                else
                {
                        out.println(HtmlString);
                }
        }
}

%>

<%
}
catch (PNException e)

{
        System.out.println("catch ERROR");
}
%>
```

Passing User Credentials/Login

This example shows how to provide login capabilities using a standard HTML form to get the information and store it in cookies. This information is used to parse a ICA template file using the TemplateParser objects. It consists of four documents: the ICA file (template_5.ica), the actual parsing file (launch_5.jsp), and a JSP page that iterates through the appEnumeration object and assigns hotlinks to each application in the collection (applaunch_5.jsp). The URL is http://%webpath%/login_5.htm. The URL for the JSP example is http://%webpath%/applaunch_5.jsp.

(LOGIN_5.HTM) Code

```
<HTML>
<HEAD>
<TITLE>NFuse Classic Example</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<!-- This is a simple HTML form that submits a username, domain and password directly to applaunch_5.asp using the HTTP POST method.-->
<form method="POST" action="applaunch_5.jsp" name="NFuseForm">
User:
<input type="text" name="user">
Domain:
<input type="text" name="domain">
Password:
<input type="password" name="password">
<input type=submit>
</form>
</body>
</html>
```

(APPLAUNCH_5.JSP) Code

```
<%@ page import = "com.citrix.nfuse.*" %>

<%
try
{
%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<HTML>
<HEAD>
        <TITLE>NFuse Classic Example</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
```

```
<%
// Declare local variables.
Credentials credentials = null;
CitrixWireGateway gateway = null;
Cookie nfuseCookie = null;
AppEnumerator appEnumerator = null;
App currentApp,app = null;
String currentFolder,folderUp,nextFolderInSelected,nfuseCookieStr,user,domain,password;

// If the username posted is from the login form, set the credentials cookie.
if  (request.getParameter("user") != null && request.getParameter("user").length() > 0)
{       // Store the login information in a cookie.
        user =  request.getParameter("user");
        password =  request.getParameter("password");
        domain =  request.getParameter("domain");
        nfuseCookieStr="NFuse_User=" + user;
        nfuseCookieStr = nfuseCookieStr + "&NFuse_Domain=" + domain;
        nfuseCookieStr = nfuseCookieStr + "&NFuse_Password=" +password;
        nfuseCookie = new Cookie("NFuseData", nfuseCookieStr);
      response.addCookie(nfuseCookie);
}
else
{
        int i1, i2;
        Cookie cookies[];
        cookies = request.getCookies();
        nfuseCookieStr = "";
        for (int i=0;i<cookies.length;i++)
        {
                if (cookies[i].getName() == null)
                {
                        response.sendRedirect("login_5.jsp");
                }
                else
                {
                        if (cookies[i].getName().equalsIgnoreCase("NFuseData"))
                {
                        if (cookies[i].getValue().length() > 0)
                                {
                        nfuseCookieStr = cookies[i].getValue();
                }
                }
                }
   }
   i1 = nfuseCookieStr.indexOf("=");
   i2 = nfuseCookieStr.indexOf("&", i1);
   user = nfuseCookieStr.substring(i1+1, i2);

   i1 = nfuseCookieStr.indexOf("=", i2);
   i2 = nfuseCookieStr.indexOf("&", i1);
   domain = nfuseCookieStr.substring(i1+1, i2);

   i1 = nfuseCookieStr.indexOf("=", i2);
   i2 = nfuseCookieStr.indexOf("&", i1);
```

```
    if (i2 == -1)
    {
        i2 = nfuseCookieStr.length();
    }

    password = nfuseCookieStr.substring(i1+1, i2);
}


// Call the App object; this allows access to methods needed later (unUrlEncode).
app = new App();

// Set credentials to represent the Credentials Java object, then initialize
// credentials with username, domain, and password.
credentials = new Credentials();
credentials.initialize
(request.getParameter("user"),request.getParameter("domain"),request.getParameter("password"));

// Set gateway to represent the Wire Gateway Java object, then initialize the gateway
// using the credentials object.
gateway = new CitrixWireGateway();
gateway.initialize(credentials);

// Set appEnumerator to establish getAppEnumerator object from the gateway object.
appEnumerator = gateway.getAppEnumerator();

// Pull the requested current folder from the user's QueryString.
currentFolder = request.getParameter("currentFolder");

// The current folder needs to be un-URL encoded. We rely on a property of the
// App object to decode this properly.
currentFolder = app.unUrlEncode(currentFolder);

// See if anything is stored in currentFolder. If not, assume the current folder is the root (null).
if (currentFolder == null || currentFolder.length() == 0)
{
        currentFolder = "";
}
else
{
        // If there is a current folder (not at root), we need to determine the parent folder.
        // This is done by parsing the currentFolder variable, taking everything prior to the last backslash.
      folderUp = currentFolder.substring(0,currentFolder.lastIndexOf("\\"));

        // The parent folder is then referenced by a link marked UP, to allow up-one-level navigation.
        // If clicked, it will refresh the page, using the parent folder as the current folder
        // using a QueryString submission.
      out.println ("<A HREF='applaunch_5.jsp?currentFolder=" + folderUp + "'>UP</A>");
      out.println ("<HR>");
}

// Iterate through the current folders using a While loop, ending when all the subfolders have been found.
while (appEnumerator.hasMoreFolders(currentFolder))
{
```

```
        nextFolderInSelected = appEnumerator.nextFolder(currentFolder);
        // The folder clicked on will refresh the page, using the selected subfolder as the
        // new current folder via QueryString submission.
        out.println ("<A HREF='applaunch_5.jsp?currentFolder=" + currentFolder + "\\");
        out.println (nextFolderInSelected + ">" + nextFolderInSelected + "</A>");
        out.println ( "<BR>");
}
out.println("<HR>");

// This While loop gives a text listing of the names of all applications in the current folder.
while (appEnumerator.hasMoreApps(currentFolder))
{
        currentApp = appEnumerator.nextApp(currentFolder);

        // When writing the application name, we enclose it in an HTML link tag. This points to
        // launch_5.jsp, with the current app name passed as a query string argument.
        // The app name, when used as part of a link, needs to be called as getNameUrlEncoded,
        // to prevent non-URL characters from invalidating the request.
        out.println ("<A HREF=launch_5.jsp?NFuse_Application=" + currentApp.getNameUrlEncoded() );
        out.println (">" + currentApp.getFriendlyName() + "</A><BR>");
}
%>
</BODY>
</HTML>


<%
}
catch (Exception e)
{
        System.out.println("catch ERROR");
}
%>
```

(TEMPLATE_5.ICA) Code

```
[WFClient]
Version=2

[ApplicationServers]
[NFuse_AppName]=

[[NFuse_AppName]]
Address=[NFuse_IPV4Address]
InitialProgram=#[NFuse_AppName]
TransportDriver=TCP/IP
WinStationDriver=ICA 3.0

AutologonAllowed=ON
[NFuse_Ticket]
[NFuse_IcaWindow]
```

(LAUNCH_5.JSP) Code

```
<%@ page import = "com.citrix.nfuse.*" %>
<%
try{
%>


<%

// Declare local variables.
TemplateParser parser;
String CookStr;
String queryString;
String HtmlString;
String UrlSessionFields;
String CurrentTransPath,cookStr;
Cookie[] cookies;

// Create the TemplateParser object, used to parse through template_1.ica and pass
// the resulting ICA file to the client.
parser = new TemplateParser();

// As in the above examples, the user, domain, and password are embedded directly
// into the file. In normal operation, pass this credential information via some other
// method (HTTP POST, cookie storage, database storage, etc.).
cookies = request.getCookies();
for (int i=0;i<cookies.length;i++)
{
        if (cookies[i].getName().equalsIgnoreCase("NFuseData"))
        {
        parser.setCookieSessionFields(cookies[i].getValue());
        }
}


// Pull all information from the user's querystring. This info is used by the TemplateParser
// to determine what application and what settings to apply; in this example, it is only used to
// set the value of NFuse_Application. See the applaunch_5.jsp file below:
// it passes this info via a link to launch_5.jsp?NFuse_Application=<appname>.
queryString = request.getQueryString();

// Set the session fields.
parser.setUrlSessionFields(queryString);

// The statement below uses JSP server variables to determine the complete path of
// this JSP file (launch_5.jsp).
CurrentTransPath = config.getServletContext().getRealPath(request.getServletPath());


// This removes the filename, leaving a complete path to the current directory.
// This path is then sent to the parser object's NFuse_TemplatesDir field.
```

```
// This directory path is used to locate the template ICA file.
if (CurrentTransPath.indexOf('\\') == -1)
{
        CurrentTransPath = CurrentTransPath.substring(0,CurrentTransPath.lastIndexOf('/'));
}
else
{
        CurrentTransPath = CurrentTransPath.substring(0,CurrentTransPath.lastIndexOf('\\'));
}

// The name of the template ICA file is then put into the NFuse_Template field.
parser.setSingleSessionField("NFuse_TemplatesDir",CurrentTransPath);
parser.setSingleSessionField("NFuse_Template", "template_5.ica");

// if parser.Parse() returns FALSE, we know that an error has occurred, and present it to the user.
if (!parser.Parse())
{
        out.println("There was an error:<br><i>" + parser.getLastError() + "</i>");
}
else
{
        // Otherwise, set the ContentType to application/x-ica (the MIME type for the Citrix ICA Client).
        response.setHeader("Content-Type", "application/x-ica");
        boolean Continue = true;
        HtmlString = "";
        // Until the parser returns EOF (a blank line), pass the current data block to the client.
        while (Continue)
        {
                HtmlString = parser.getNextDataBlock();
                if (HtmlString.length() == 0)
                {
                        Continue = false;
                }
                else
                {
                        out.println(HtmlString);
                }
        }
}

%>

<%
}
catch (PNException e)

{
        System.out.println("catch ERROR");
}
%>
```

Caching

This example demonstrates how caching works. Applications are cached and used to parse a ICA template file using the TemplateParser objects. It consists of three documents: the ICA file (template_6.ica), the actual parsing file (launch_6.jsp), and a JSP page that iterates through the appEnumeration object and assigns hotlinks to each application in the collection. The URL for this example is http://%webpath%/applaunch_6.jsp.

(APPLAUNCH_6.JSP) Code

```
<%@ page import = "com.citrix.nfuse.*" %>
<!-- Create a cache object to store the AppDataList objects -->
<jsp:useBean id="citrixCache" scope="application" class="com.citrix.nfuse.AppListCache" />

<%
try
{
%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<HTML>
<HEAD>
        <TITLE>NFuse Classic Example</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<%
// Declare local variables.
Credentials credentials = null;
CitrixWireGateway gateway = null;
AppEnumerator appEnumerator = null;
App currentApp,app = null;
AppDataList apps = null;
String currentFolder,folderUp,nextFolderInSelected,key,user,domain,password;

// First call the App object; this allows access to methods needed later (unUrlEncode).
app = new App();

// Set credentials to represent the Credentials Java object, then initialize
// credentials with username, domain, and password.
credentials = new Credentials();
user = "user";
domain = "domain";
password = "password";

credentials.initialize (user, domain, password);

// Set gateway to represent the Wire Gateway Java object, then initialize the gateway
// using the credentials object.
gateway = new CitrixWireGateway();
gateway.initialize(credentials);
```

```
// Create a name for this object stored in the cache.
key = app.unUrlEncode(domain) + "\\" +  app.unUrlEncode(user) + "\\" + app.unUrlEncode(password);
key = key.toUpperCase();

// Try to get the data from the cache first.
if (citrixCache.contain(key)) {
   apps = citrixCache.retrieveFromCache(key);
}

// If the data is not in the cache or it is expired, get it from the wire.
if (apps == null || apps.isExpired()) {
   apps = gateway.getAppDataList();
}

if (apps != null) {
   // Add into the cache.
   citrixCache.addToCache(key, apps);
}

// Set appEnumerator to establish getAppEnumerator object.
appEnumerator = apps.getAppEnumerator();

// Pull the requested current folder from the user's QueryString.
currentFolder = request.getParameter("currentFolder");

// The current folder needs to be un-URL encoded. We rely on a property of the
// App object to decode this properly.
currentFolder = app.unUrlEncode(currentFolder);

// See if anything is stored in currentFolder. If not, assume the current folder is the root (null).
if (currentFolder == null || currentFolder.length() == 0)
{
        currentFolder = "";
}
else
{
        // If there is a current folder (not at root), we need to determine the parent folder.
        // This is done by parsing the currentFolder variable, taking everything prior to the last backslash.
      folderUp = currentFolder.substring(0,currentFolder.lastIndexOf("\\"));

        // The parent folder is then referenced by a link marked UP, to allow up-one-level navigation.
        // If clicked, it will refresh the page, using the parent folder as the current folder
        // via QueryString submission.
      out.println ("<A HREF='applaunch_6.jsp?currentFolder=" + folderUp + "'>UP</A>");
      out.println ("<HR>");
}

// Iterate through the current folders via a While loop, ending when all the subfolders have been found.
while (appEnumerator.hasMoreFolders(currentFolder))
{
        nextFolderInSelected = appEnumerator.nextFolder(currentFolder);
        // The folder clicked on will refresh the page, using the selected subfolder as the
        // new current folder via QueryString submission.
```

```
        out.println ("<A HREF='applaunch_6.jsp?currentFolder=" + currentFolder + "\\");
        out.println (nextFolderInSelected + ">" +nextFolderInSelected + "</A>");
        out.println ( "<BR>");
}
out.println("<HR>");

// This While loop gives a text listing of the names of all applications in the current folder.
while (appEnumerator.hasMoreApps(currentFolder))
{
        currentApp = appEnumerator.nextApp(currentFolder);

        // When writing the application name, we enclose it in an HTML link tag. This points to
        // launch_6.jsp, with the current app name passed as a query string argument.
        // The app name, when used as part of a link, needs to be called as getNameUrlEncoded,
        // to prevent non-URL characters from invalidating the request.
        out.println ("<A HREF=launch_6.jsp?NFuse_Application=" + currentApp.getNameUrlEncoded() );
        out.println (">" + currentApp.getFriendlyName() + "</A><BR>");
}
%>
</BODY>
</HTML>


<%
}
catch (Exception e)
{
        System.out.println("catch ERROR");
}
%>
```

(TEMPLATE_6.ICA) Code

```
[WFClient]
Version=2

[ApplicationServers]
[NFuse_AppName]=

[[NFuse_AppName]]
Address=[NFuse_IPV4Address]
InitialProgram=#[NFuse_AppName]
TransportDriver=TCP/IP
WinStationDriver=ICA 3.0

AutologonAllowed=ON
[NFuse_Ticket]
[NFuse_IcaWindow]
```

(LAUNCH_6.JSP) Code

```
<%@ page import = "com.citrix.nfuse.*" %>
<%
try{
```

```
%>


<%

// Declare local variables.
TemplateParser parser;
String CookStr;
String queryString;
String HtmlString;
String UrlSessionFields;
String CurrentTransPath;
Cookie[] cookies;

// Create the TemplateParser object, used to parse through template_6.ica and pass
// the resulting ICA file to the client.
parser = new TemplateParser();

// As in the above examples, the user, domain, and password are embedded directly
// into the file. In normal operation, pass this credential information via some other
// method (HTTP POST, cookie storage, database storage, etc.).
parser.setCookieSessionFields("NFuse_User=username&NFuse_Password=password&NFuse_Domain=domain");

// Pull all information from the user's querystring. This info is used by the TemplateParser
// to determine what application and what settings to apply; in this example, it is only used to
// set the value of NFuse_Application. See the applaunch_6.jsp file below:
// it passes this info via a link to launch_6.jsp?NFuse_Application=<appname>.
queryString = request.getQueryString();

// Set the session fields.
parser.setUrlSessionFields(queryString);

// The statement below uses JSP server variables to determine the complete path of
// this JSP file (Launch_6.jsp).
CurrentTransPath = config.getServletContext().getRealPath(request.getServletPath());


// This removes the filename, leaving a complete path to the current directory.
// This path is then sent to the parser object's NFuse_TemplatesDir field.
// This directory path is used to locate the template ICA file.
if (CurrentTransPath.indexOf('\\') == -1)
{
        CurrentTransPath = CurrentTransPath.substring(0,CurrentTransPath.lastIndexOf('/'));
}
else
{
        CurrentTransPath = CurrentTransPath.substring(0,CurrentTransPath.lastIndexOf('\\'));
}

// The name of the template ICA file is then put into the NFuse_Template field.
parser.setSingleSessionField("NFuse_TemplatesDir",CurrentTransPath);
parser.setSingleSessionField("NFuse_Template", "template_6.ica");

// if parser.Parse() returns FALSE, an error has occurred, and it is presented to the user.
```

```
if (!parser.Parse())
{
	out.println("There was an error:<br><i>" + parser.getLastError() + "</i>");
}
else
{
	// Otherwise, set the ContentType to application/x-ica (the MIME type for the Citrix ICA Client).
	response.setHeader("Content-Type", "application/x-ica");
	boolean Continue = true;
	HtmlString = "";
	// Until the parser returns EOF (a blank line), pass the current data block to the client.
	while (Continue)
	{
		HtmlString = parser.getNextDataBlock();
		if (HtmlString.length() == 0)
		{
			Continue = false;
		}
		else
		{
			out.println(HtmlString);
		}
	}
}

%>

<%
}
catch (PNException e)

{
	System.out.println("catch ERROR");
}
%>
```

APPENDIX A

Citrix® NFuse™ Classic Software License Agreement

This is a legal agreement ("AGREEMENT") between you, the Licensed User or representative of the Licensed User and Citrix Systems, Inc., or Citrix Systems International GmbH. Citrix Systems, Inc., a Delaware corporation, markets and supports this Citrix NFuse™ product (hereinafter "PRODUCT") in the Americas. Citrix Systems International GmbH, a Swiss company wholly owned by Citrix Systems, Inc., markets and supports this PRODUCT in Europe, the Middle East, Africa, Asia and the Pacific. Your location of receipt of this PRODUCT determines which is the licensing entity hereunder (the applicable entity is hereinafter referred to as "CITRIX"). BY OPENING THE SEALED DISK PACKAGE OR DOWNLOADING THE PRODUCT, YOU ARE AGREEING TO BE BOUND BY THE TERMS OF THIS AGREEMENT. IF YOU DO NOT AGREE TO THE TERMS OF THIS AGREEMENT, PROMPTLY RETURN THE UNUSED DISK PACKAGE TO THE PLACE WHERE YOU OBTAINED IT FOR A FULL REFUND OR EXIT THE DOWNLOAD PROCESS TO AVOID CHARGES.

1. GRANT OF LICENSE. This PRODUCT contains software that provides services on a computer called a server (”Server Software”), and may contain software that allows a computer to access or utilize the services provided by the Server Software (”Client Software”). CITRIX grants to you the following non-exclusive rights to the Server Software and Client Software, if any, and accompanying documentation (collectively called the ”SOFTWARE”):

a) **Installation and Transfer**. You may install the Server Software on a single web server (the ”Server”). You may also install the Client Software on as many computers, operating remotely or on one or more networks, as your needs require (the "Client(s)"), provided that these computers are used to access the Server, but the number of concurrent logins will be limited pursuant to Section 1.b.

b) **Use of the Server Software**. You may use one copy of the Server Software at any time on the Server for support of MetaFrame™ or MetaFrame XP™ clients. The Server Software supports usage by more than one user at a time, but it may be used to support only up to the number of user logins you are entitled to based on your purchase of CITRIX MetaFrame™ or MetaFrame XP™licenses. Any attempt to use the SOFTWARE in violation of these limitations is a breach of this AGREEMENT. You may make one (1) copy of the SOFTWARE in machine readable form solely for back-up purposes, provided that you reproduce all proprietary notices on the copy.

c) **Use of the Client Software**. You may use the Client Software to access the Server.

d) **Other**. **Notice to Users** - You shall inform all users of the SOFTWARE of the terms and conditions of this AGREEMENT. **Not For Resale Software** - If this SOFTWARE is labeled ”Not For Resale” or ”NFR,” your license only permits use for demonstration, test, or evaluation purposes.

2. DESCRIPTION OF OTHER RIGHTS AND LIMITATIONS. For all SOFTWARE - You may use the Server Software for the provision of application services to third party customers ("Hosting"). You may not otherwise rent or lease the SOFTWARE, but you may transfer the SOFTWARE and accompanying written materials on a permanent basis, provided you retain no copies and the recipient agrees to the terms of this AGREEMENT. You may use the scripting capability to customize the look, feel and functionality of the Server Software. You may not otherwise modify, translate, reverse engineer, decompile, or disassemble, create derivative works based on, or copy (except for back-up as permitted above) the SOFTWARE, except to the extent such foregoing restriction is expressly prohibited by applicable law. You may not remove any proprietary notices, labels, or marks on the SOFTWARE.

YOU MAY NOT USE, COPY, MODIFY, OR TRANSFER THE SOFTWARE OR ANY COPY IN WHOLE OR IN PART, OR GRANT ANY RIGHTS IN THE SOFTWARE OR ACCOMPANYING DOCUMENTATION, EXCEPT AS EXPRESSLY PROVIDED IN THIS AGREEMENT. ALL RIGHTS NOT EXPRESSLY GRANTED ARE RESERVED BY CITRIX OR ITS SUPPLIERS.

You hereby agree, that to the extent that any applicable mandatory laws (such as, for example, national laws implementing EC Directive 91/250 on the Legal Protection of Computer Programs) give you the right to perform any of the aforementioned activities without CITRIX's consent in order to gain certain information about the SOFTWARE for purposes specified in the respective statutes, before you exercise any such rights, you shall first request such information from CITRIX in writing detailing the purpose for which you need the information. Only if and after CITRIX, at its sole discretion, partly or completely denies your request, shall you exercise your statutory rights.

Limited Warranty and Disclaimer. CITRIX warrants that, for a period of ninety (90) days from the date of delivery of the SOFTWARE to you, the media on which the SOFTWARE is furnished, if any, under normal use will be free from defects in materials and workmanship, and that the SOFTWARE will perform substantially in accordance with the CITRIX product documentation published by CITRIX and included with the SOFTWARE. CITRIX and its suppliers' entire liability and your exclusive remedy under this warranty (which is subject to you returning the SOFTWARE to CITRIX or an authorized reseller) will be, at CITRIX's option, to replace the media and/or SOFTWARE or to refund the purchase price and terminate this AGREEMENT.

EXCEPT FOR THE ABOVE EXPRESS LIMITED WARRANTIES, CITRIX AND ITS SUPPLIERS MAKE AND YOU RECEIVE NO WARRANTIES OR CONDITIONS, EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE, AND CITRIX AND ITS SUPPLIERS SPECIFICALLY DISCLAIM ANY CONDITIONS OF QUALITY AND ANY IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, OR FITNESS FOR A PARTICULAR PURPOSE. YOU ASSUME THE RESPONSIBILITY FOR THE SELECTION OF THE SOFTWARE AND HARDWARE TO ACHIEVE YOUR INTENDED RESULTS, AND FOR THE INSTALLATION OF, USE OF, AND RESULTS OBTAINED FROM THE SOFTWARE AND HARDWARE. CITRIX DOES NOT WARRANT THAT USE OF THE SOFTWARE WILL BE UNINTERRUPTED OR ERROR FREE.

Proprietary Rights. This license is not a sale. Title and copyright rights to the SOFTWARE, accompanying documentation and any copies made by you remain with CITRIX or its suppliers.

Limitation of Liability. TO THE EXTENT PERMITTED BY APPLICABLE LAW, YOU AGREE THAT NEITHER CITRIX NOR ITS AFFILIATES, SUPPLIERS OR AUTHORIZED DISTRIBUTORS SHALL BE LIABLE FOR ANY LOSS OF DATA, LOSS OF INCOME, LOSS OF OPPORTUNITY OR PROFITS, COST OF RECOVERY OR ANY OTHER SPECIAL, INCIDENTAL, CONSEQUENTIAL OR INDIRECT DAMAGES ARISING OUT OF OR IN CONNECTION WITH THIS AGREEMENT, OR THE USE OF THE SOFTWARE, REFERENCE MATERIALS OR ACCOMPANYING DOCUMENTATION, HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY. THIS LIMITATION WILL APPLY EVEN IF CITRIX, ITS AFFILIATES, SUPPLIERS OR AUTHORIZED DISTRIBUTORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN NO EVENT SHALL THE LIABILITY OF CITRIX, ITS AFFILIATES, SUPPLIERS OR AUTHORIZED DISTRIBUTORS EXCEED THE AMOUNT PAID FOR THE LICENSED SOFTWARE AT ISSUE. YOU ACKNOWLEDGE THAT THE LICENSE FEE REFLECTS THIS ALLOCATION OF RISK. SOME JURISDICTIONS DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY TO YOU. For purposes of this Agreement, the term "CITRIX AFFILIATE" shall mean any legal entity fifty percent (50%) or more of the voting interests in which are owned directly or indirectly by Citrix Systems, Inc. Affiliates, suppliers and authorized distributors are intended to be third party beneficiaries of this AGREEMENT.

Export Restriction. You agree that you will not export or re-export the SOFTWARE in any form without the appropriate government licenses. Your failure to comply with this provision is a material breach of this AGREEMENT.

Termination. This AGREEMENT is effective until terminated. You may terminate this AGREEMENT at any time by removing the SOFTWARE from your computer and destroying all copies. Unauthorized copying of the SOFTWARE or the accompanying documentation or otherwise failing to comply with the terms and conditions of this AGREEMENT will result in automatic termination of this license and will make available to CITRIX other legal remedies. Upon termination of this AGREEMENT, the license granted herein will terminate and you must immediately destroy the SOFTWARE and accompanying documentation, and all back-up copies thereof.

Government End-Users. If you are a U.S. Government agency, in accordance with Section 12.212 of the Federal Acquisition Regulation (48 CFR 12.212), you hereby acknowledge that use, duplication and disclosure of the SOFTWARE by the U.S. Government or any of its agencies is governed by, and subject to, all of the terms, conditions, restrictions and limitations set forth in this AGREEMENT. In the event that, for any reason, Section 12.212 is not applicable, you hereby acknowledge that use, duplication and disclosure of the SOFTWARE by U.S. Government agencies is subject to the Commercial Computer Software Restricted Rights clause at 48 CFR Section 52.227-19(c)(1) and (2), or the Rights in technical Data and Computer Software clause at DFARS 252.227-7013, as applicable. Manufacturer is Citrix Systems, Inc., 6400 Northwest Sixth Way, Fort Lauderdale, Florida, 33309.

If licensor is Citrix Systems, Inc., this AGREEMENT will be governed by the laws of the State of Florida without reference to conflict of laws principles and excluding the United Nations Convention on Contracts for the International Sale of Goods, and in any dispute arising out of this AGREEMENT, you consent to the exclusive personal jurisdiction and venue in the State and Federal courts within Broward County, Florida. If licensor is Citrix Systems International GmbH, this AGREEMENT will be governed by the laws of Switzerland without reference to the conflict of laws principles, and excluding the United Nations Convention on Contracts for the International Sale of Goods, and in any dispute arising out of this AGREEMENT, you consent to the exclusive personal jurisdiction and venue of the competent courts in the Canton of Zürich.

Should you have any questions concerning this AGREEMENT, or wish to contact licensor for any reason, please write to licensor at the following address: Citrix Systems, Inc., Customer Service, 6400 Northwest Sixth Way, Fort Lauderdale, Florida, 33309; or Citrix Systems International GmbH, Rheinweg 9, CH-8200 Schaffhausen, Switzerland.

Citrix is a registered trademark of Citrix Systems, Inc., in the U.S. and other countries.