



eServices 8.0

Web API Client

Developer's Guide

The information contained herein is proprietary and confidential and cannot be disclosed or duplicated without the prior written consent of Genesys Telecommunications Laboratories, Inc.

Copyright © 2003–2011 Genesys Telecommunications Laboratories, Inc. All rights reserved.

About Genesys

Alcatel-Lucent's Genesys solutions feature leading software that manages customer interactions over phone, Web, and mobile devices. The Genesys software suite handles customer conversations across multiple channels and resources—self-service, assisted-service, and proactive outreach—fulfilling customer requests and optimizing customer care goals while efficiently using resources. Genesys software directs more than 100 million customer interactions every day for 4000 companies and government agencies in 80 countries. These companies and agencies leverage their entire organization, from the contact center to the back office, while dynamically engaging their customers. Go to www.genesyslab.com for more information.

Each product has its own documentation for online viewing at the Genesys Technical Support website or on the Documentation Library DVD, which is available from Genesys upon request. For more information, contact your sales representative.

Notice

Although reasonable effort is made to ensure that the information in this document is complete and accurate at the time of release, Genesys Telecommunications Laboratories, Inc., cannot assume responsibility for any existing errors. Changes and/or corrections to the information contained in this document may be incorporated in future versions.

Your Responsibility for Your System's Security

You are responsible for the security of your system. Product administration to prevent unauthorized use is your responsibility. Your system administrator should read all documents provided with this product to fully understand the features available that reduce your risk of incurring charges for unlicensed use of Genesys products.

Trademarks

Genesys, the Genesys logo, and T-Server are registered trademarks of Genesys Telecommunications Laboratories, Inc. All other trademarks and trade names referred to in this document are the property of other companies. The Crystal monospace font is used by permission of Software Renovation Corporation, www.SoftwareRenovation.com.

Technical Support from VARs

If you have purchased support from a value-added reseller (VAR), please contact the VAR for technical support.

Technical Support from Genesys

If you have purchased support directly from Genesys, please contact Genesys Technical Support at the regional numbers provided on [page 19](#). For complete contact information and procedures, refer to the [Genesys Technical Support Guide](#).

Ordering and Licensing Information

Complete information on ordering and licensing Genesys products can be found in the [Genesys Licensing Guide](#).

Released by

Genesys Telecommunications Laboratories, Inc. www.genesyslab.com

Document Version: 80mm_dev_web-api_04-2011_v8.0.211.00



Table of Contents

List of Procedures	11
Preface	13
eServices and the CIM Platform	13
CIM Platform	14
eServices	14
About eServices Web API and Samples	16
Intended Audience	16
Usage Guidelines	16
Making Comments on This Document	18
Contacting Genesys Technical Support	19
Document Change History	19
New in Document Version 8.0.101.00	19
New in Document Version 8.0.102.00	21
New in Document Version 8.0.201.00	22
New in Document Version 8.0.202.00	22
Chapter 1	
About Web API Clients	25
Architecture	25
Packages	26
Packets and Envelopes	26
API Accessibility (Java)	27
Configuration Server	27
Load Balancing	28
Balancing Multiple Web API Servers	29
International Language Support	33
API Usage in the Samples	34
About Web API Server	43
Chapter 2	
About the Samples	45
Overview	45
Samples	46

	Test Tools	48
	Installing the Samples	48
	Tools You Need Before Installation	48
	Installation Process	48
	Installation Testing	48
	Configuring for Statistical Information	48
	Directory Structure	49
	Java	49
	.NET	49
	Sample Files	50
	Test Tool Files	58
Chapter 3	Understanding and Using the E-Mail Service	61
	Overview	61
	Life Cycle of an E-Mail Session	62
	Event Flow of an E-Mail Session	62
Chapter 4	Understanding and Using the Flex Chat Service	65
	Overview	65
	Life Cycle of a Chat Session	66
	Event Flow of a Chat Session	67
	Transcripts	69
Chapter 5	Understanding and Using the Genesys 3rd Party Media Service	71
	Overview	71
	Architecture	72
	Event Flow of a Request	72
Chapter 6	Understanding and Using the Web Collaboration Service	75
	What is Cobrowsing?	75
	Architecture	76
	Web Collaboration Process	76
	Integrating Cobrowsing into Your Application	78
Chapter 7	Understanding and Using the FAQ Service	81
	Overview	81
	Genesys Knowledge Management	82
	Genesys Content Analyzer	82
	FAQ Objects	83

	Sample FAQ.jar File	83
Chapter 8	Understanding and Using the Web Callback Service	85
	Overview.....	85
	Architecture	85
	Web Callback API.....	86
	WCBRequestSubmit.....	87
	WCBRequestCancel.....	90
	WCBRequestReschedule	93
	WCBRequestGetSingleInteraction	96
	WCBRequestGetMultipleInteractions	98
Chapter 9	eServices Samples for Java	101
	Overview.....	102
	Sample Overview.....	102
	Shared Files	104
	File Descriptions	105
	E-Mail Sample	108
	Purpose	109
	Functionality Overview.....	109
	Files	109
	Code Explanation	110
	E-Mail with Attachments Sample.....	113
	Purpose	113
	Functionality Overview.....	113
	Files	114
	Code Explanation	114
	Chat Sample.....	121
	Purpose	121
	Functionality Overview.....	121
	Files	122
	Code Explanation	122
	Advanced Chat.....	132
	Purpose	133
	Functionality Overview.....	133
	Files	133
	Code Explanation	133
	Chat Widget Sample.....	134
	Purpose	134
	Functionality Overview.....	134
	Files	135
	Code Explanation	135

Survey Sample	138
Purpose	138
Functionality Overview	138
Files	139
Code Explanation	139
Web Callback Sample	144
Purpose	144
Functionality Overview	145
Files	145
Code Explanation	145
Interaction Submit (Genesys 3rd Party Media) Sample	151
Purpose	152
Functionality Overview	152
Files	153
Code Explanation	153
Statistical Information Sample	160
Purpose	160
Functionality Overview	160
Files	160
Code Explanation	160
Universal Contact Server Sample	166
Purpose	166
Functionality Overview	166
Files	167
Code Explanation	167
Additional Customizations	180
Dynamic Invitation Sample	181
Purpose	181
Functionality Overview	181
Files	182
Code Explanation	182
FAQ (Web Self-Serve) Sample	187
Purpose	187
Functionality Overview	187
Files	187
Code Explanation	188
Media Availability Sample	193
Purpose	193
Functionality Overview	193
Files	194
Code Explanation	194
Cobrowse Samples Overview	197
Common Files	197

Basic Cobrowse Sample	197
Purpose	197
Functionality Overview	198
Files	198
Code Explanation	198
Chat and Cobrowse Sample	203
Purpose	203
Functionality Overview	203
Files	203
Code Explanation	204
Cobrowse with Meet Me	211
Purpose	211
Functionality Overview	211
Files	211
Code Explanation	211
Cobrowse with Initial Startup Page	214
Purpose	214
Functionality Overview	214
Files	215
Code Explanation	215
Cobrowse with Dynamic Startup Page Sample	218
Purpose	218
Functionality Overview	218
Files	218
Code Explanation	219
Build Your Own Dynamic Startup Page Example	221
 Chapter 10	
eServices Samples for .NET	225
Overview	225
Samples Included	226
Files Included: .ASPX Versus .ASPX.CS	228
Shared Files	228
File Descriptions	229
Web Callback Sample	230
Purpose	230
Functionality Overview	230
Files	231
Code Explanation	231
Chat Sample	238
Purpose	238
Functionality Overview	238
Files	238
Code Explanation	238

Chat with AJAX Sample	251
Purpose	251
Functionality Overview	251
Files	252
Code Explanation	252
Survey Sample	262
Purpose	262
Functionality Overview	262
Files	262
Code Explanation	263
E-Mail Sample	269
Purpose	269
Functionality Overview	270
Files	270
Code Explanation	270
Genesys 3rd Party Media Sample	274
Purpose	275
Functionality Overview	275
Files	276
Code Explanation	276
Stat Server Sample	286
Purpose	286
Functionality Overview	286
Files	286
Code Explanation	286
Universal Contact Server Sample	294
Purpose	294
Functionality Overview	295
Files	295
Code Explanation	295
Cobrowse Samples Overview	308
Common Files	308
Basic Cobrowse Sample	309
Purpose	309
Functionality Overview	309
Files	309
Code Explanation	309
Chat and Cobrowse Sample	315
Purpose	315
Functionality Overview	315
Files	315
Code Explanation	316
Cobrowse with Meet Me	322

	Purpose	323
	Functionality Overview	323
	Files	323
	Code Explanation	323
	Cobrowse with Initial Startup Page	326
	Purpose	326
	Functionality Overview	326
	Files	326
	Code Explanation	327
	Cobrowse with Dynamic Startup Page Sample	330
	Purpose	330
	Functionality Overview	330
	Files	330
	Code Explanation	331
Chapter 11	eServices Test Tools	335
	Overview	335
	Java	335
	.NET	336
	Using the Tools	337
	Load-Balancing Servlet Configuration	337
	Verifying Chat Server Configuration	338
	Verifying the Configuration of E-Mail Server	339
	Verifying Stat Server Configuration	340
	Troubleshooting Guide	341
Chapter 12	Sample Client Scenarios	343
	Disclaimers	343
	Common Scenarios	343
	Scenario 1	344
	Scenario 2	344
	Scenario 3	345
	Scenario 4	345
	Scenario 5	346
	Conclusion	346
Appendix A	Updating Your Customized Code	347
	Overview	347
	Common Code	347
	Chat Sample	348
	Code Comparisons	348

	E-Mail Sample	352
	Code Comparisons	352
	Interaction Submit Sample	353
	Code Comparisons	353
Appendix B	Statistical Information Sample Configuration	359
	The webapistats.cfg File	359
Supplements	Related Documentation Resources	363
	Document Conventions	366
Index		369



List of Procedures

Using the eServices Load Balancing API to balance multiple instances of Web API Server	29
Using the eServices Load Balancing API to balance multiple instances of Chat Server	31
Using the Load Balancing API to balance multiple instances of E-Mail Server	33



Preface

Welcome to the *eServices 8.0 Web API Client Developer's Guide*. This document will show you how to turn your call center into an Internet contact center by adding a website that offers chat and e-mail support. It will also show you how to use Genesys 3rd Party Media services to submit web-based interactions of virtually any type to Genesys Interaction Server.

This document is valid only for the 8.0 release(s) of this product.

Note: For versions of this document created for other releases of this product, visit the Genesys Technical Support website, or request the Documentation Library DVD, which you can order by e-mail from Genesys Order Management at orderman@genesyslab.com.

This preface contains the following sections:

- [eServices and the CIM Platform, page 13](#)
- [About eServices Web API and Samples, page 16](#)
- [Intended Audience, page 16](#)
- [Usage Guidelines, page 16](#)
- [Making Comments on This Document, page 18](#)
- [Contacting Genesys Technical Support, page 19](#)
- [Document Change History, page 19](#)

For information about related resources and about the conventions that are used in this document, see the supplementary material starting on [page 363](#).

eServices and the CIM Platform

The Genesys eServices Web API gives you the programming tools you need to write client applications that use chat and e-mail services.

eServices (formerly Multimedia) is a cover term for Genesys components that work together to manage interactions whose media is something other than traditional telephonic voice (for example, e-mail or chat).

eServices includes some parts of the Genesys Customer Interaction Management (CIM) Platform, plus certain of the media channels that run on top of the Platform.

CIM Platform

The CIM Platform consists of the following:

- Management Framework
- Reporting (CC Analyzer, CCPulse+)
- Interaction Management, which in turn consists of:
 - Universal Routing
 - Interaction Workflow
 - Knowledge Management
 - Content Analysis
 - Universal Contact History

On top of the CIM Platform are various media channels. Some, such as Genesys Network Voice, handle traditional telephony. Others, such as Genesys E-mail, handle other media.

eServices

eServices, then, consists of the following:

- From the CIM Platform, all of Interaction Management except for Universal Routing:
 - Interaction Workflow—centralized handling of interactions irrespective of media type
 - Knowledge Management—creation and maintenance of standard responses and screening rules
 - Content Analysis—optional enhancement to Knowledge Management, applying natural language processing technology to categorize interactions
 - Universal Contact History—storage of data on contacts and on interactions (linked as threads)

Universal Routing is not considered part of eServices because it deals with both traditional telephonic interactions and the nontraditional interactions that are handled in eServices.

- From the media channels, at least one of the following:
 - Genesys E-mail—e-mail
 - Genesys Web Media—chat

- Genesys 3rd Party Media—ability to add customized support for other media (fax, for example)
- Optionally, Web Collaboration—the ability for agents and customers to cobrowse (simultaneously navigate) shared web pages. This is an option that you can add to either Genesys Web Media or Inbound Voice.

Figure 1 shows the Genesys components that make up eServices.

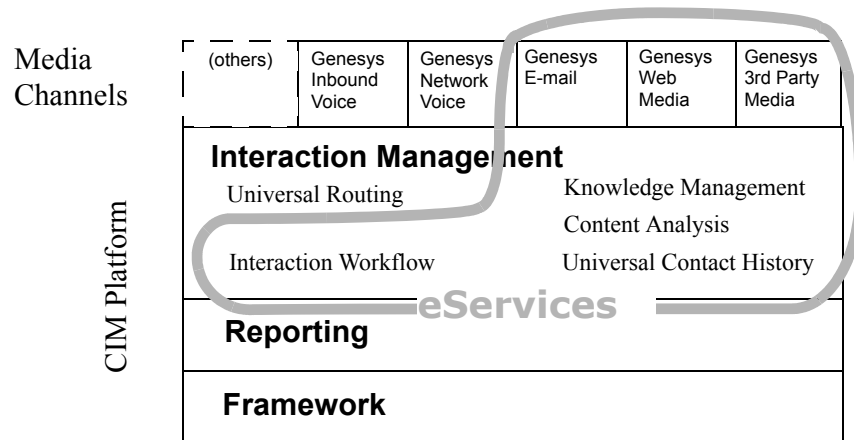


Figure 1: eServices in Relation to the CIM Platform and Media Channels

Note: Although Universal Routing is not considered part of eServices, any functioning solution (platform plus channels) that includes any part of the Interaction Management sector requires Universal Routing.

Each component has its own documentation. For details about obtaining documents, see “Related Documentation Resources” on [page 363](#).

Licensing

Licensing requirements are:

- For each agent: one eServices Agent seat.
- For each media option: one media channel (E-mail and/or Web Media and/or custom media).
- For Genesys Content Analyzer: NLP Content Analysis license.

See also the *Genesys Licensing Guide*.

Reporting

Reporting templates are available for eServices. For details, see the *Reporting Technical Reference Guide for the Genesys 8.x Release*.

About eServices Web API and Samples

The eServices Web API enables you to write client applications that support chat, e-mail, web collaboration, and Frequently Asked Questions (FAQ) interactions. It includes predefined, media-specific packages for these purposes.

The eServices Sample applications demonstrate a particular eServices Web API feature or a combination of features.

Intended Audience

This document is primarily intended for developers. It has been written with the assumption that you have a basic understanding of:

- Computer-telephony integration (CTI) concepts, processes, terminology, and applications
- Network design and operation
- Your own network configurations

You should also be familiar with:

- Genesys Framework architecture and functions
- Genesys eServices architecture and functions
- Java servlet technologies, including JavaServer Pages (JSPs)
- For the .NET implementation: the Microsoft .NET Framework and Active Server Pages (ASPXs)
- Java and JavaScript, or comparable scripting languages like VB Script and Jscript
- Hypertext Markup Language (HTML)
- Underlying technologies that support web servers and servlet engines

Usage Guidelines

The Genesys developer materials outlined in this document are intended to be used for the following purposes:

- Creation of contact center agent desktop applications associated with Genesys software implementations.
- Server-side integration between Genesys software and third-party software.
- Creation of a specialized client application specific to customer needs.

The Genesys software functions available for development are clearly documented. No undocumented functionality is to be utilized without the express written consent of Genesys.

The following Use Conditions apply in all cases for developers employing the Genesys developer materials outlined in this document:

1. Possession of interface documentation does not imply a right to use by a third party. Genesys conditions for use, as outlined below or in the *Genesys Developer Program Guide*, must be met.
2. This interface shall not be used unless the developer is a member in good standing of the Genesys Interacts program or has a valid Master Software License and Services Agreement with Genesys.
3. A developer shall not be entitled to use any licenses granted hereunder unless the developer's organization has met or obtained all prerequisite licensing and software as set out by Genesys.
4. A developer shall not be entitled to use any licenses granted hereunder if the developer's organization is delinquent in any payments or amounts owed to Genesys.
5. A developer shall not use the Genesys developer materials outlined in this document for any general application development purposes that are not associated with the above-mentioned intended purposes for the use of the Genesys developer materials outlined in this document.
6. A developer shall disclose the developer materials outlined in this document only to those employees who have a direct need to create, debug, and/or test one or more participant-specific objects and/or software files that access, communicate, or interoperate with the Genesys API.
7. The developed works and Genesys software running in conjunction with one another (hereinafter referred to together as the "integrated solutions") should not compromise data integrity. For example, if both the Genesys software and the integrated solutions can modify the same data, then modifications by either product must not circumvent the other product's data integrity rules. In addition, the integration should not cause duplicate copies of data to exist in both participant and Genesys databases, unless it can be assured that data modifications propagate all copies within the time required by typical users.
8. The integrated solutions shall not compromise data or application security, access, or visibility restrictions that are enforced by either the Genesys software or the developed works.
9. The integrated solutions shall conform to design and implementation guidelines and restrictions described in the *Genesys Developer Program Guide* and Genesys software documentation. For example:
 - a. The integration must use only published interfaces to access Genesys data.

- b. The integration shall not modify data in Genesys database tables directly using SQL.
- c. The integration shall not introduce database triggers or stored procedures that operate on Genesys database tables.

Any schema extension to Genesys database tables must be carried out using Genesys Developer software through documented methods and features.

The Genesys developer materials outlined in this document are not intended to be used for the creation of any product with functionality comparable to any Genesys products, including products similar or substantially similar to current Genesys general-availability, beta, and announced products.

Any attempt to use the Genesys developer materials outlined in this document or any Genesys Developer software contrary to this clause shall be deemed a material breach with immediate termination of this addendum, and Genesys shall be entitled to seek to protect its interests, including but not limited to, preliminary and permanent injunctive relief, as well as money damages.

Making Comments on This Document

If you especially like or dislike anything about this document, feel free to e-mail your comments to Techpubs.webadmin@genesyslab.com.

You can comment on what you regard as specific errors or omissions, and on the accuracy, organization, subject matter, or completeness of this document. Please limit your comments to the scope of this document only and to the way in which the information is presented. Contact your Genesys Account Representative or Genesys Technical Support if you have suggestions about the product itself.

When you send us comments, you grant Genesys a nonexclusive right to use or distribute your comments in any way it believes appropriate, without incurring any obligation to you.

Contacting Genesys Technical Support

If you have purchased support directly from Genesys, contact Genesys Technical Support at the following regional numbers:

Region	Telephone	E-Mail
North America and Latin America	+888-369-5555 (toll-free) +506-674-6767	support@genesyslab.com
Europe, Middle East, and Africa	+44-(0)-1276-45-7002	support@genesyslab.co.uk
Asia Pacific	+61-7-3368-6868	support@genesyslab.com.au
Malaysia	1-800-814-472 (toll-free) +61-7-3368-6868 (International)	support@genesyslab.com.au
India	000-800-100-7136 (toll-free) +61-7-3368-6868 (International)	support@genesyslab.com.au
Japan	+81-3-6361-8950	support@genesyslab.co.jp
Before contacting technical support, refer to the <i>Genesys Technical Support Guide</i> for complete contact information and procedures.		

Document Change History

This section lists topics that are new or that have changed significantly since the first release of this document.

New in Document Version 8.0.101.00

[Table 1](#) provides details about what is new or has changed significantly from version 8.0.001.00 to 8.0.101.00 of this document.

Table 1: Document Changes

Heading/Topic	Details
Chapter 1, “About Web API Clients,” on page 25	
Table 5, “API Access by Samples,” on page 35 .	Information added for the new Survey, Chat Widget, and Web Callback Samples.
Table 6, “eServices Samples,” on page 46 .	Information added for the new Survey, Chat Widget, and Web Callback Samples.
Chapter 2, “About the Samples,” on page 45	
Chapter 2, “Directory Structure,” on page 49 .	Information added for the new Survey, Chat Widget, Web Callback, and Cobrowse Samples.
Chapter 8, “Understanding and Using the Web Callback Service,” on page 85	
Chapter 8, “Understanding and Using the Web Callback Service,” on page 85 .	Information regarding the new Web Callback Sample.
Chapter 9, “eServices Samples for Java,” on page 101	
“Chat Widget Sample” on page 134 .	Java code snippets for the new Chat Widget Sample.
“Survey Sample” on page 138 .	Java code snippets for the new Survey Sample.
“Web Callback Sample” on page 144 .	Java code snippets for the new Web Callback Sample.
Chapter 10, “eServices Samples for .NET,” on page 225	
“Web Callback Sample” on page 230 .	.NET code snippets for the new Web Callback Sample.
“Cobrowse Samples Overview” on page 308 .	Overview for the new .NET Cobrowse Samples.
“Basic Cobrowse Sample” on page 309 .	.NET code snippets for the new Basic Cobrowse Sample.
“Chat and Cobrowse Sample” on page 315 .	.NET code snippets for the new Chat and Cobrowse Sample.
“Cobrowse with Meet Me” on page 322 .	.NET code snippets for the new Cobrowse with Meet Me Sample.

Table 1: Document Changes (Continued)

Heading/Topic	Details
“Cobrowse with Initial Startup Page” on page 326 .	.NET code snippets for the new Cobrowse with Initial Startup Page Sample.
“Cobrowse with Dynamic Startup Page Sample” on page 330 .	.NET code snippets for the new Cobrowse with Dynamic Startup Page.

New in Document Version 8.0.102.00

[Table 2](#) provides details about what is new or has changed significantly from version 8.0.101.00 to 8.0.102.00 of this document.

Table 2: Document Changes

Heading/Topic	Details
Chapter 9, “eServices Samples for Java,” on page 101	
Table 12, “Constant Values in the constants.jsp File,” on page 105 .	New values added for Web Callback Sample.
“Getting Load Balancer, Interaction Server, and Creating the Protocol Object” on page 145 .	New Java code snippet.
“Create the Web Callback Object” on page 146 .	New Java code snippet.
Chapter 10, “eServices Samples for .NET,” on page 225	
“Getting Load Balancer, Interaction Server, and Creating the Protocol Object” on page 231 .	New .NET code snippet.
“Create the Web Callback Object” on page 232 .	New .NET code snippet.

New in Document Version 8.0.201.00

[Table 3](#) provides details about what is new or has changed significantly from version 8.0.102.00 to 8.0.201.00 of this document.

Table 3: Document Changes

Heading/Topic	Details
The entire guide.	All virtual routes were updated to reflect 8.0.2 instead of 8.0.1.
Chapter 9, “eServices Samples for Java,” on page 101 .	Updated code snippets for Email samples. and Email with Attachments samples.
Chapter 10, “eServices Samples for .NET,” on page 225 .	Updated code snippets for Email samples.
Chapter 10, “eServices Samples for .NET,” on page 225 .	Added section for Survey samples.

New in Document Version 8.0.211.00

[Table 4](#) provides details about what is new or has changed significantly from version 8.0.201.00 to 8.0.211.00 of this document.

Table 4: Document Changes

Heading/Topic	Details
The entire guide.	References to voice callback were removed.
Chapter 9, “eServices Samples for Java,” on page 101 .	Updated code snippets for the following samples: • Chat • Advanced Chat • Chat Widget • Chat and CoBrowse • Basic CoBrowse • CoBrowse with Meet Me • CoBrowse with Dynamic Startup Page • Dynamic Invitation • E-mail • E-mail with Attachments • FAQ (Web Self-Serve) • Interaction Submit (Genesys 3rd Party Media) • Media Availability • Statistical Information • Survey • Universal Contact Server • Web Callback
Chapter 10, “eServices Samples for .NET,” on page 225 .	Updated code snippets for the following samples: • Chat • Chat with Ajax • Chat and CoBrowse • Basic CoBrowse • CoBrowse with Dynamic Startup Page • E-mail • Genesys 3rd Party Media • Stat Server • Survey • Universal Contact Server • Web Callback



Chapter

1

About Web API Clients

The eServices Web API enables you to write client applications that support chat, e-mail, web-collaboration, and FAQ interactions. It includes predefined, media-specific packages for these purposes. By using Genesys 3rd Party Media, you can also write applications that handle custom media types.

This chapter provides the context that you need to make practical use of the Web API. It contains the following sections:

- [Architecture, page 25](#)
- [Packets and Envelopes, page 26](#)
- [API Accessibility \(Java\), page 27](#)
- [API Usage in the Samples, page 34](#)
- [About Web API Server, page 43](#)

Architecture

The Web API enables your customized server pages (JSPs and ASPXs) to perform common functions such as sending an e-mail message or conducting a chat session. These common functions are performed via communication with a servlet container—such as WebSphere, WebLogic, or Tomcat—that is running on a web server. The servlet container processes the JSP or ASPX and forwards the content to the appropriate server. [Figure 2](#) shows this process for an e-mail message.

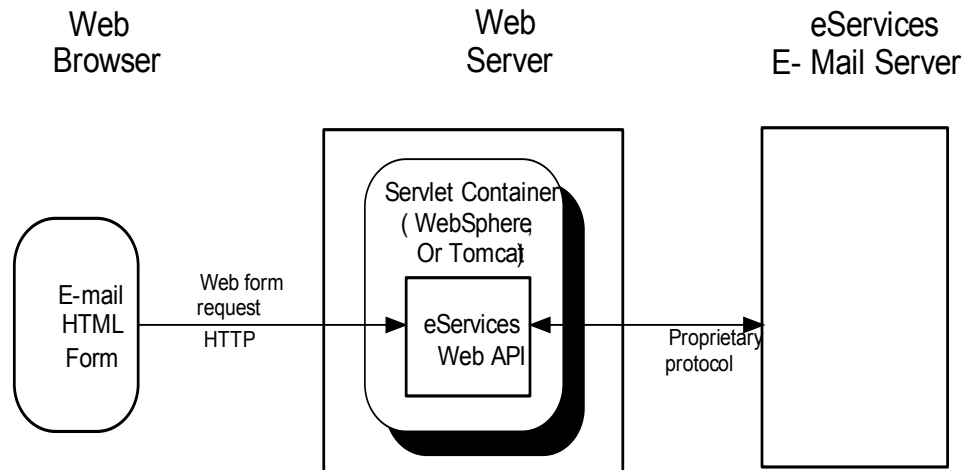


Figure 2: Sending an E-Mail Message to eServices (Container Could Also Be WebLogic)

Packages

The Web API's Java and .NET implementations are similar and contain a single “flat” directory. The directory contains a large number of Platform SDK packages, which appear in the following list:

- Platform SDK Configuration
- Platform SDK Contacts
- Platform SDK Management
- Platform SDK Open Media
- Platform SDK Outbound
- Platform SDK Statistics
- Platform SDK Voice
- Platform SDK Web Media
- Platform SDK Routing (.NET only)

For detailed information about Platform SDK, see the:

- *Platform SDK 8.0 Deployment Guide*, which contains important configuration and installation information.
- *Platform SDK 8.0 API Reference* (for the particular SDK that you are using), which provides the authoritative information on methods, functions, and events for your SDK.

Packets and Envelopes

Genesys 3rd Party Media uses interaction-based processing to transmit binary data. The other packages of the Web API use XML. The following discussion

will help you understand the XML-based packages, which were designed using the metaphor of *packets* and *envelopes*.

A packet is a collection of data. An envelope holds the packet. An *application* creates a packet and places it inside an envelope, which is produced by an *envelope factory*. The application then passes the envelope around, in the system. [Figure 3](#) illustrates this concept.

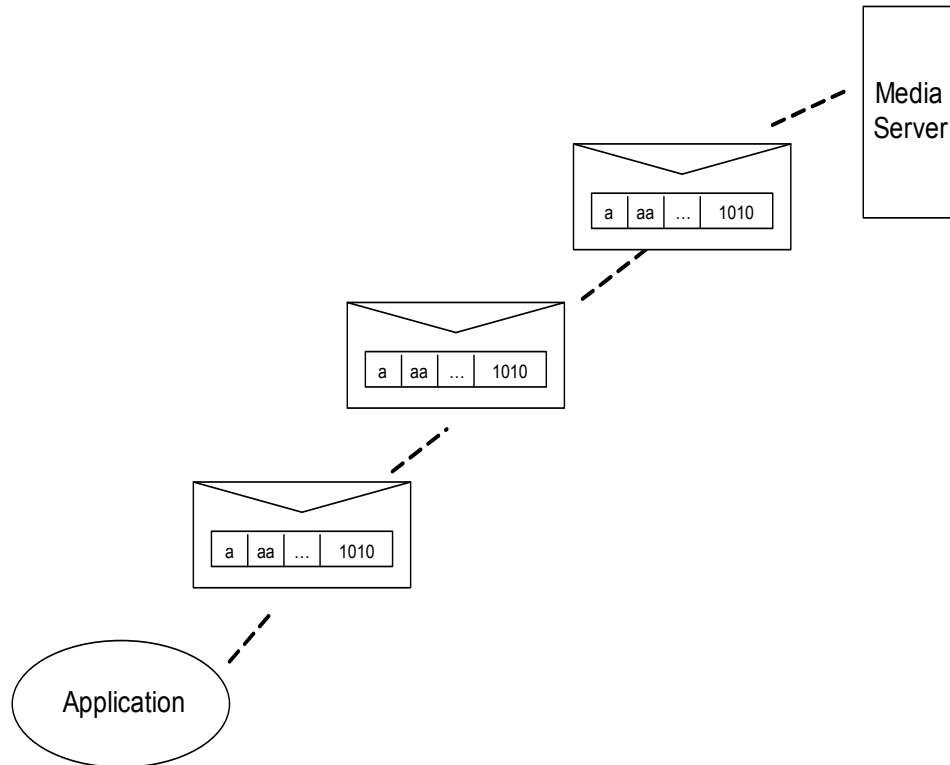


Figure 3: Packets and Envelopes

API Accessibility (Java)

This section explains how the Java API retrieves the data and configuration options from Genesys Configuration Server to make the sample applications work properly.

Configuration Server

The Web API must access Configuration Server and retrieve certain option settings; otherwise, the samples will not work properly.

Web Sample and Configuration Access

The “Interaction Submit (Genesys 3rd Party Media) Sample” on [page 151](#) retrieves data with regard to available destinations for interaction from

Configuration Server. The endpoints:NNN section of the Web API Server application is read and available destinations are added to the list that is used by the sample. For more information, review the `ItxSubmit.jsp` file and everything that relates to the `kvcAppEndPoints` variable.

For direct access to Configuration Server options, use the following method from the `com.genesyslab.webapi.core.LoadBalancer` class:

```
public static String getOption (String strAppName, String strSectionName,
    String strOptionName, String strDefaultValue) throws LoadBalancerException.
```

The following methods are useful for communicating with Configuration Server.

To retrieve the tenant DBID by tenant name, use the following method:

```
public static String getTenantId(String strTenantName) throws LoadBalancerException;
```

To retrieve the Configuration Server application that represents Web API Server .NET, use the following method:

```
public static ConfigServerApp getOwnApp() throws LoadBalancerException;
```

To retrieve Configuration Server options from a list, use the following method:

```
public static String getOptionFromList(KeyValueCollection kvcOptions,
    String strSectionName, String strOptionName, String strDefaultValue)
```

To retrieve the Configuration Server application by application name, use the following method:

```
public static ConfigServerApp getConfigServerApp(String strAppName) throws
    LoadBalancerException;
```

Test-Tool Access

The eServices test tools include two files—`ChatSelfTest.jsp` and `EmailSelfTest.jsp`—that access the Configuration Server during their diagnostic tests. These files retrieve the options from the Chat Server and E-mail Server Application objects, respectively. Chapter 11 on [page 335](#) discusses the test tools in more detail.

Load Balancing

The Web API allows you take advantage of eServices's load balancing features. This section briefly describes these features and how to use them. For more detailed information, refer to the *eServices 8.0 User's Guide*.

Load balancing provides greater scalability and service availability by providing multiple instances of certain eServices servers. The redundancy that

is provided by load balancing also helps prevent loss of data. Load balancing can take place within a tenant or across tenants.

The Load Balancing API provides the following functionality:

- It can select a particular server instance from a set of instances of the specified server type.
- Upon the first request to a server instance, the API can create an alias for the selected server instance and store it for future use.
- It can use the alias to obtain connection parameters (host name and port) of the server instance.
- It has access to configuration information.

Balancing Multiple Web API Servers

Web API Server runs on your web server, which is the front end of the Internet contact center. To handle high volumes of incoming messages or requests, you might want to run multiple web servers, and therefore, also run multiple instances of Web API Server. This section describes load balancing by one Web API Server among other instances of Web API Server.

The actual required number of Web API Servers depends on several factors, such as host performance, required number of simultaneous chat sessions, chat update interval (see “Limitations” on [page 32](#)), and network performance.

Note: More sophisticated third-party solutions are available to balance traffic among multiple server instances. You can use one of these alternatives instead of the Load Balancing API that this section describes.

Procedure:

Using the eServices Load Balancing API to balance multiple instances of Web API Server

Purpose: To use the eServices Load Balancing API to load balance data traffic across multiple instances of Web API Server.

Prerequisites

- Configure and install several instances of Web API Server.
- Designate one of the instances of Web API Server for load balancing, and configure connections from it to all other instances of Web API Server in the configuration.

You can use Application Clusters to simplify configuration, as this procedure describes.

Note: For load balancing to work, servers must have a connection to Solution Control Server (SCS).

Start of procedure

1. Create an instance of the `ServiceInfo` class.

Note: Use this instance of the class for only one thread. Additional threads require additional instances.

2. Use the `LoadBalancer.GetServiceInfo(CfgAppType.CFGServerType, strTenant)` method to select an instance of Web API Server.
3. Use the `getHost()` and `getWebApiPort()` methods to produce redirect instructions. The `getAlias()` method can be used to get the alias name.
4. If you know the alias of a specific server instance and want to be able to connect to that instance again, you must use the `LoadBalancer.GetServiceInfo(strAliasName)` method.

End of procedure

Code Sample and Explanation

Declare an instance of the `ServiceInfo` class; then use the `GetServiceInfo()` method for your first request when you do not know the alias of the server, as the following code sample shows:

```
ServiceInfo si = null;
try
{
    si = LoadBalancer.GetServiceInfo(CfgAppType.CFGWebAPIServer,
        strTenant);
}
```

Use the `getHost()`, `getWebApiPort()`, and `getAlias()` methods to obtain host and port numbers, as well as the alias name:

```
String svcHost = si.getHost();
int svcPort = si.getWebApiPort();
String svcAliasName = si.getAlias();
}
```

If an exception is thrown, display an appropriate message to the user:

```
catch (LoadBalancerException e)
{
    //Add message here
}
```

Use the following code for your second request when you know the alias of the server and want to connect to it again:

```
try
{
    si = LoadBalancer.GetServiceInfo(svcAliasName);
    svcHost = si.getHost();
    svcPort = si.getWebApiPort();
}
```

If an exception is thrown, display an appropriate message to the user:

```
catch (LoadBalancerException e)
{
    //Add message here
}
```

Note: The host in which Web API Server is running must have access to Configuration Server, SCS, and Message Server (optional).

Procedure:

Using the eServices Load Balancing API to balance multiple instances of Chat Server

Purpose: An overview of the load balancing procedure that is used for Chat Servers.

Because chat interactions take place online, only the Chat Server that handles a particular chat session can process requests about that session. This requires that the web application select the Chat Server while the chat session is being established, and that it continue to use the same Chat Server for all subsequent requests.

Prerequisites

- For load balancing by Web API Server between multiple Chat Servers, you must configure multiple applications of the Chat Server type. Web API Server must have either direct connections to Chat Servers or connections to Chat Servers through Application Clusters.

Start of procedure

1. For every user request, create an instance of the `ServiceInfo` class.
It is possible to use only one instance, if it is thread-safe.

2. When the user starts the chat session:
 - a. Call the `LoadBalancer.GetServiceInfo(CfgAppType.CFGChatServer, strTenant)` method to select the Chat Server for that session.
 - b. Call the `getAlias()` method to obtain the alias of the selected server alias and store it in the application. The alias of the server is transferred to and from the client part of the web application.
3. For all subsequent requests as the chat session continues, call the `LoadBalancer.GetServiceInfo(<chat server alias>)` method to obtain the Chat Server by its stored alias.
4. For any request, use the `getHost()` and `getPort()` methods to obtain the Chat Server host and port.

End of procedure

Sample

The HTML chat sample uses a separate frame for communication with Chat Server and consists of the following files:

- `HtmlChatFrameSet.jsp` (Java) or `ChatFrameset.htm` (.NET)—The sample main frame.
- `HtmlChatPanel.jsp` (Java) or `ChatPanel.aspx` (.NET)—The content of the frame that contains the chat form and is visible to the user. The page is requested once from Web API Server.
- `HtmlChatCommand.jsp` (Java) or `ChatCommand.aspx` (.NET)—The content of the invisible frame that interacts with the Chat Server and makes any necessary changes to the visible frame.

Limitations

- While you are developing a chat application by using the eServices Load Balancing API, you must ensure that all subsequent requests with regard to a chat session are directed to the same Chat Server that established the chat session. Additional Chat Servers cannot process these requests.
- You must set a constant for the update time interval. Do so by using the `chatRefreshTimeout` variable in the `...WebAPISamples80\constants.jsp` file. Be aware that if this time interval is too long, the application will seem slow to the user. On the other hand, if the time interval is too short, the Web API Servers and Chat Servers will have to bear an unnecessarily high load. A shorter interval might require more instances of Web API Server, as “Using the eServices Load Balancing API to balance multiple instances of Web API Server” on [page 29](#) describes.
- A good rule of thumb is that the shorter the update time interval, the more Web API Servers are needed.

Procedure: Using the Load Balancing API to balance multiple instances of E-Mail Server

The HTML e-mail sample consists of the `Email.jsp` (Java) or `Email.aspx` (.NET) file.

Note: Communication with E-mail Server for web-form submission is stateless. So, you might prefer not to store an alias for E-mail Server and, instead select a new instance for every web-form submission.

Purpose: To use the eServices Load Balancing API to load balance data traffic across multiple instances of E-mail Server.

Prerequisites

- For load balancing among multiple instances of E-mail Server, you must configure multiple applications of the E-mail Server type. Web API Server must have either direct connections to instances of E-mail Server or connections to instances of E-mail Server through Application Clusters.

Start of procedure

1. Gathers information from the web-form submission.
2. Gets e-mail host and port from the Load Balancing API. The operations in this step are identical to those in “Using the eServices Load Balancing API to balance multiple instances of Web API Server” on [page 29](#).
3. Prepares data for the eServices Web API.
4. Submits the request via the eServices Web API.
5. Returns the `RefID` of the request—possibly, with an error code and description.

End of procedure

International Language Support

eServices supports multiple natural languages for the client applications. However, client requests must be in the same natural language as the media server with which the client is interacting. To resolve this, an international language support class—`IntSupport`—facilitates interaction between the client and server data.

For example, you might have a client who is using an Asian language, such as Chinese or Japanese, that uses a multibyte character set (MBCS). Your customized server pages (JSPs or ASPXs) must use this character set when

they interact with this client. The `HttpServletResponse` class has a method called `setContentType()`, which your server pages can use to do just that. You can choose the character set on the `Options` tab under the `Application` object of the Web API Server.

Likewise, the media servers send Unicode data. The servlet (and engine) must respond to the client in the appropriate character set, which in this multibyte case is not Unicode. However, this can occur only if the server knows both the correct character set and the correct code page for this data, because the same combination of MBCS characters might have different Unicode values for different code pages. For example, you cannot use the `charset` option for Chinese characters and the `code page` option for Cyrillic characters at the same time.

The following code snippet shows how a client who is using a different natural language can communicate with the server. The client gets the correct response based on the charset that is set by the JSP or ASPX:

```
<%response.setContentType("text/html; charset="+ i18nsupport.GetCharSet());;%>
```

The `GetCharSet()` method invokes internal methods that set the code page and charset options correctly before it returns the value. The default code page is Microsoft's Cp1252, and the default character set is ISO-8859-1.

The following line from the sample JSP file retrieves the first name of a user from the `i18nsupport` class instead of from the `HttpServletRequest` class. The `GetSubmitParametr()` method retrieves the requested parameters and processes them. In case of internationalization, the method returns a different character-code value, which is properly handled by the customized server page. For clients that have the same code value as the server, the following line does not change anything.

```
String first_name = i18nsupport.GetSubmitParametr(request, fldnFirstName);
```

API Usage in the Samples

The samples use various parts of the eServices Web API to accomplish their tasks.

Note: Use this section in conjunction with the *Platform SDK 8.0 Java (or .NET) API Reference*. These are located on the Genesys Developer Documentation Library DVD. They are also installed with the software, at the following default location:

`<Web_API_Server_application_object_name>\doc`. These references give details on how to use the classes and their methods. For an explanation of how to create a sample application, see Chapter 9 on [page 101](#).

Each sample uses one or more specific APIs. Table 5 on [page 35](#) lists their API usage.

Table 5: API Access by Samples

Sample	Web APIs Used
E-mail	genesyslab.webapi.core.ServiceInfo genesyslab.webapi.core.LoadBalancer genesyslab.webapi.core.LoadBalancerException genesyslab.webapi.utils.i18n.i18nsupport genesyslab.platform.configuration.protocol.types.CfgAppType genesyslab.platform.common.protocol.Message genesyslab.platform.common.protocol.Endpoint genesyslab.platform.webmedia.protocol.EspEmailProtocol genesyslab.platform.webmedia.protocol.EspEmail genesyslab.platform.webmedia.protocol.EspEmail.requests genesyslab.platform.webmedia.protocol.EspEmail.events
E-mail with Attachment	genesyslab.platform.common.protocol.Endpoint genesyslab.platform.common.protocol.Message genesyslab.platform.webmedia.protocol.EspEmailProtocol genesyslab.platform.webmedia.protocol.EspEmail.* genesyslab.platform.webmedia.protocol.EspEmail.events.* genesyslab.platform.webmedia.protocol.EspEmail.requests.* genesyslab.platform.configuration.protocol.types.CfgAppType genesyslab.webapi.core.LoadBalancer genesyslab.webapi.core.LoadBalancerException genesyslab.webapi.core.ServiceInfo genesyslab.webapi.utils.i18n.i18nsupport genesyslab.webapi.utils.base64.base64

Table 5: API Access by Samples (Continued)

Sample	Web APIs Used
Chat	genesyslab.webapi.core.ServiceInfo genesyslab.webapi.core.LoadBalancer genesyslab.webapi.utils.i18n.* genesyslab.webapi.utils.i18n.i18nsupport genesyslab.platform.configuration.protocol.types.CfgAppType genesyslab.platform.common.protocol.Endpoint genesyslab.platform.common.collections.KeyValueCollection genesyslab.platform.common.protocol.Message genesyslab.platform.webmedia.protocol.* genesyslab.platform.webmedia.protocol.flexchat.requests.RequestLogin genesyslab.platform.webmedia.protocol.flexchat.events.EventStatus genesyslab.platform.webmedia.protocol.flexchat.requests.RequestJoin genesyslab.platform.webmedia.protocol.flexchat.requests.RequestRefresh genesyslab.platform.webmedia.protocol.flexchat.requests.RequestLogout genesyslab.platform.webmedia.protocol.flexchat.* genesyslab.platform.openmedia.protocol.interactionserver.PartyInfo
Advanced Chat	genesyslab.webapi.core.ServiceInfo genesyslab.webapi.core.LoadBalancer genesyslab.webapi.utils.i18n.* genesyslab.webapi.utils.i18n.i18nsupport genesyslab.platform.configuration.protocol.types.CfgAppType genesyslab.platform.common.protocol.Endpoint genesyslab.platform.common.protocol.Message genesyslab.platform.common.collections.KeyValueCollection genesyslab.platform.webmedia.protocol.* genesyslab.platform.webmedia.protocol.flexchat.* genesyslab.platform.webmedia.protocol.flexchat.requests.RequestLogin genesyslab.platform.webmedia.protocol.flexchat.events.EventStatus genesyslab.platform.webmedia.protocol.flexchat.requests.RequestJoin genesyslab.platform.webmedia.protocol.flexchat.requests.RequestRefresh genesyslab.platform.webmedia.protocol.flexchat.requests.RequestLogout genesyslab.platform.openmedia.protocol.interactionserver.PartyInfo

Table 5: API Access by Samples (Continued)

Sample	Web APIs Used
Genesys 3rd Party Media (Interaction Submit)	genesyslab.webapi.core.ServiceInfo genesyslab.webapi.core.LoadBalancer genesyslab.webapi.core.LoadBalancerException genesyslab.webapi.core.ConfigServerApp genesyslab.webapi.utils.i18n.i18nsupport genesyslab.platform.common.collections.KeyValueCollection genesyslab.platform.common.collections.KeyValuePair genesyslab.platform.common.collections.ValueType genesyslab.platform.common.protocol.Endpoint genesyslab.platform.common.protocol.Message genesyslab.platform.openmedia.protocol.InteractionServerProtocol genesyslab.platform.openmedia.protocol.interactionserver.requests.interactionmanagement.RequestSubmit genesyslab.platform.openmedia.protocol.interactionserver.events.EventAck genesyslab.platform.openmedia.protocol.interactionserver.events.EventError genesyslab.platform.openmedia.protocol.interactionserver.requests.interactionmanagement.RequestChangeProperties genesyslab.platform.openmedia.protocol.interactionserver.ReasonInfo genesyslab.platform.openmedia.protocol.interactionserver.requests.interactionmanagement.RequestStopProcessing genesyslab.platform.openmedia.protocol.interactionserver.InteractionClient genesyslab.platform.configuration.protocol.types.CfgAppType
Basic Cobrowse	genesyslab.webapi.utils.i18n.i18nsupport genesyslab.webapi.core.LoadBalancer genesyslab.webapi.core.ServiceInfo genesyslab.platform.configuration.protocol.types.CfgAppType
Cobrowse with Dynamic Startup Page	genesyslab.webapi.utils.i18n.i18nsupport genesyslab.webapi.core.LoadBalancer genesyslab.webapi.core.ServiceInfo genesyslab.platform.configuration.protocol.types.CfgAppType
Cobrowse with Initial Startup Page	genesyslab.webapi.utils.i18n.i18nsupport genesyslab.webapi.core.LoadBalancer genesyslab.webapi.core.ServiceInfo genesyslab.platform.configuration.protocol.types.CfgAppType

Table 5: API Access by Samples (Continued)

Sample	Web APIs Used
Cobrowse with Meet Me	genesyslab.webapi.utils.i18n.i18nsupport genesyslab.webapi.core.LoadBalancer genesyslab.webapi.core.ServiceInfo genesyslab.platform.configuration.protocol.types.CfgAppType
Chat and Cobrowse	genesyslab.webapi.core.ServiceInfo genesyslab.webapi.core.LoadBalancer genesyslab.webapi.utils.i18n.* genesyslab.webapi.utils.i18n.i18nsupport genesyslab.platform.configuration.protocol.types.CfgAppType genesyslab.platform.common.protocol.Endpoint genesyslab.platform.common.collections.KeyValueCollection genesyslab.platform.common.protocol.Message genesyslab.platform.webmedia.protocol.* genesyslab.platform.webmedia.protocol.flexchat.requests.RequestLogin genesyslab.platform.webmedia.protocol.flexchat.events.EventStatus genesyslab.platform.webmedia.protocol.flexchat.requests.RequestJoin genesyslab.platform.webmedia.protocol.flexchat.requests.RequestRefresh genesyslab.platform.webmedia.protocol.flexchat.requests.RequestLogout genesyslab.platform.webmedia.protocol.flexchat.* genesyslab.platform.openmedia.protocol.interactionserver.PartyInfo
FAQ (Web Self-Serve)	genesyslab.webapi.utils.i18n.i18nsupport Genesys.webapi.media.faq.direct.*
Media Availability	genesyslab.webapi.core.ServiceInfo genesyslab.webapi.core.LoadBalancer genesyslab.webapi.core.LoadBalancerException genesyslab.webapi.utils.i18n.i18nsupport genesyslab.platform.configuration.protocol.types.CfgAppType

Table 5: API Access by Samples (Continued)

Sample	Web APIs Used
Statistical Information	genesyslab.webapi.core.ServiceInfo genesyslab.webapi.core.LoadBalancer genesyslab.webapi.core.LoadBalancerException genesyslab.webapi.core.ConfigServerApp genesyslab.webapi.utils.i18n.i18nsupport genesyslab.platform.common.collections.KeyValueCollection genesyslab.platform.common.collections.KeyValuePair genesyslab.platform.common.collections.ValueType genesyslab.platform.common.protocol.Endpoint genesyslab.platform.common.protocol.Message genesyslab.platform.common.protocol.ProtocolException genesyslab.platform.common.protocol.ChannelState genesyslab.platform.configuration.protocol.types.CfgAppType genesyslab.platform.reporting.protocol.StatServerProtocol genesyslab.platform.reporting.protocol.statserver.* genesyslab.platform.reporting.protocol.statserver.requests.RequestOpenPackage genesyslab.platform.reporting.protocol.statserver.events.EventPackageInfo genesyslab.platform.reporting.protocol.statserver.events.EventPackageOpened genesyslab.platform.reporting.protocol.statserver.events.EventPackageError

Table 5: API Access by Samples (Continued)

Sample	Web APIs Used
Universal Contact Server	genesyslab.webapi.core.ServiceInfo genesyslab.webapi.core.LoadBalancer genesyslab.webapi.core.LoadBalancerException genesyslab.webapi.core.ConfigServerApp genesyslab.platform.commons.collections.KeyValueCollection genesyslab.platform.commons.collections.KeyValuePair genesyslab.platform.commons.collections.ValueType genesyslab.platform.commons.protocol.Endpoint genesyslab.platform.commons.protocol.Message genesyslab.platform.commons.protocol.ProtocolException genesyslab.platform.commons.protocol.ChannelState genesyslab.platform.configuration.protocol.types.CfgAppType genesyslab.platform.reporting.protocol.StatServerProtocol genesyslab.platform.reporting.protocol.statserver.* genesyslab.platform.reporting.protocol.statserver.requests.RequestOpenPackage genesyslab.platform.reporting.protocol.statserver.events.EventPackageInfo genesyslab.platform.reporting.protocol.statserver.events.EventPackageOpened genesyslab.platform.reporting.protocol.statserver.events.EventPackageError genesyslab.platform.contacts.protocol.* genesyslab.platform.contacts.protocol.contactserver.requests.RequestGetInteractionsForContact genesyslab.platform.contacts.protocol.contactserver.requests.RequestUpdateInteraction genesyslab.platform.contacts.protocol.contactserver.requests.RequestGetContacts genesyslab.platform.contacts.protocol.contactserver.* genesyslab.platform.contacts.protocol.contactserver.events.EventUpdateInteraction genesyslab.platform.contacts.protocol.contactserver.events.EventGetContacts genesyslab.platform.contacts.protocol.contactserver.events.EventError genesyslab.platform.contacts.protocol.contactserver.events.EventGetInteractionsForContact genesyslab.platform.contacts.protocol.contactserver.StringList

Table 5: API Access by Samples (Continued)

Sample	Web APIs Used
Survey	genesyslab.webapi.core.ServiceInfo genesyslab.webapi.core.LoadBalancer genesyslab.webapi.core.LoadBalancerException genesyslab.webapi.core.ConfigServerApp genesyslab.webapi.utils.i18n.i18nsupport genesyslab.platform.common.collections.KeyValueCollection genesyslab.platform.common.collections.KeyValuePair genesyslab.platform.common.collections.ValueType genesyslab.platform.common.protocol.Endpoint genesyslab.platform.openmedia.protocol.interactionserver.requests.interactionmanagement.RequestSubmit genesyslab.platform.common.protocol.Message genesyslab.platform.openmedia.protocol.interactionserver.events.EventAck genesyslab.platform.openmedia.protocol.interactionserver.events.EventError genesyslab.platform.openmedia.protocol.interactionserver.requests.interactionmanagement.RequestChangeProperties genesyslab.platform.openmedia.protocol.interactionserver.ReasonInfo genesyslab.platform.openmedia.protocol.interactionserver.requests.interactionmanagement.RequestStopProcessing genesyslab.platform.openmedia.protocol.interactionserver.InteractionClient genesyslab.platform.configuration.protocol.types.CfgAppType genesyslab.platform.contacts.protocol.InteractionAttributeListConstants genesyslab.platform.contacts.protocol.contactserver.requests.RequestGetContacts genesyslab.platform.contacts.protocol.contactserver.* genesyslab.platform.contacts.protocol.contactserver.requests.RequestGetInteractionsForContact genesyslab.platform.contacts.protocol.contactserver.StringList genesyslab.platform.contacts.protocol.* genesyslab.platform.contacts.protocol.contactserver.requests.RequestUpdateInteraction genesyslab.platform.contacts.protocol.contactserver.requests.RequestGetInteractionContent genesyslab.platform.contacts.protocol.contactserver.events.*

Table 5: API Access by Samples (Continued)

Sample	Web APIs Used
Chat Widget	genesyslab.platform.webmedia.protocol.* genesyslab.webapi.core.ServiceInfo genesyslab.webapi.core.LoadBalancer genesyslab.webapi.utils.i18n.* genesyslab.platform.configuration.protocol.types.CfgAppType genesyslab.platform.common.protocol.Endpoint genesyslab.platform.common.collections.KeyValueCollection genesyslab.platform.webmedia.protocol.flexchat.requests.RequestLogin genesyslab.platform.common.protocol.Message genesyslab.platform.webmedia.protocol.flexchat.events.EventStatus genesyslab.platform.webmedia.protocol.flexchat.requests.RequestJoin genesyslab.platform.webmedia.protocol.flexchat.requests.RequestRefresh genesyslab.platform.webmedia.protocol.flexchat.requests.RequestLogout genesyslab.platform.webmedia.protocol.flexchat.* genesyslab.platform.openmedia.protocol.interactionserver.PartyInfo genesyslab.webapi.utils.i18n.i18nsupport
Web Callback	genesyslab.platform.common.collections.KeyValueCollection genesyslab.platform.common.protocol.ChannelState genesyslab.platform.common.protocol.Endpoint genesyslab.platform.common.protocol.Message genesyslab.platform.configuration.protocol.types.CfgAppType genesyslab.platform.openmedia.protocol.InteractionServerProtocol genesyslab.platform.openmedia.protocol.interactionserver.InteractionClient genesyslab.platform.openmedia.protocol.interactionserver.events.EventError genesyslab.platform.openmedia.protocol.interactionserver.requests.agentmanagement.RequestAgentLogin genesyslab.webapi.core.LoadBalancer genesyslab.webapi.core.ServiceInfo genesyslab.webapi.utils.i18n.i18nsupport genesyslab.webcallback.*

About Web API Server

Web API Server performs the following steps to connect to Configuration Server:

1. Reads the primary/backup Configuration Server information from the `web.xml` file.
2. Connects to the primary/backup Configuration Server by using warm-standby functionality.
3. Reads the ADDP information for configuring a connection to Configuration Server.

Web API Server follows a strict set of rules to determine to which Configuration Server it will connect:

Rule 1: In the connection tab of the Web API Server application there must be only one Configuration Server. If there is more than one connected Configuration Server, Web API Server will take into consideration only the first one from the list and ignore all of the others.

Rule 2: Web API Server always uses only primary/backup Configuration Server from the `web.xml` file. It ignores any Configuration Servers that have been added to the connection tab of the application except for purpose of the configuring the ADDP protocol.

Rule 3: If Web API Server does not receive notifications from Configuration Server Proxy about the loss of the connection to Configuration Server, warm standby will not reconnect to the backup Configuration Server. Warm standby reacts only to an open/close event for the connection. It does not control the status of the Configuration Server application that is reported by SCS/LCA.



Chapter

2

About the Samples

This chapter briefly describes the sample applications that come with the eServices Web API. Also, it outlines how to install them and it describes the files and directories that you must create to run these samples. Later chapters explain these components in detail.

The information in this chapter is divided among the following topics:

- [Overview, page 45](#)
- [Installing the Samples, page 48](#)
- [Directory Structure, page 49](#)

Overview

This section describes the sample applications. [Figure 4](#) shows the start page for both the java Samples and Test Tools. A similar start page exists for the .NET Samples and Test Tools.

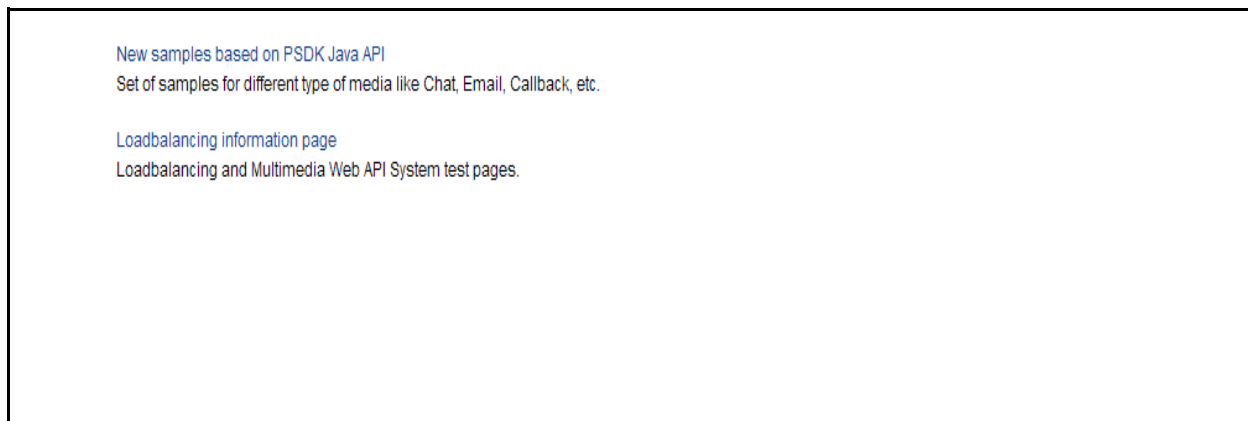


Figure 4: eServices Samples and Test Tool Start Page

Samples

Each sample application demonstrates a particular Web API feature or feature combination. [Figure 5](#) shows the main Menu of the Java Samples. From here you can access all of the Java Samples. A similar page exists for the .NET Samples.

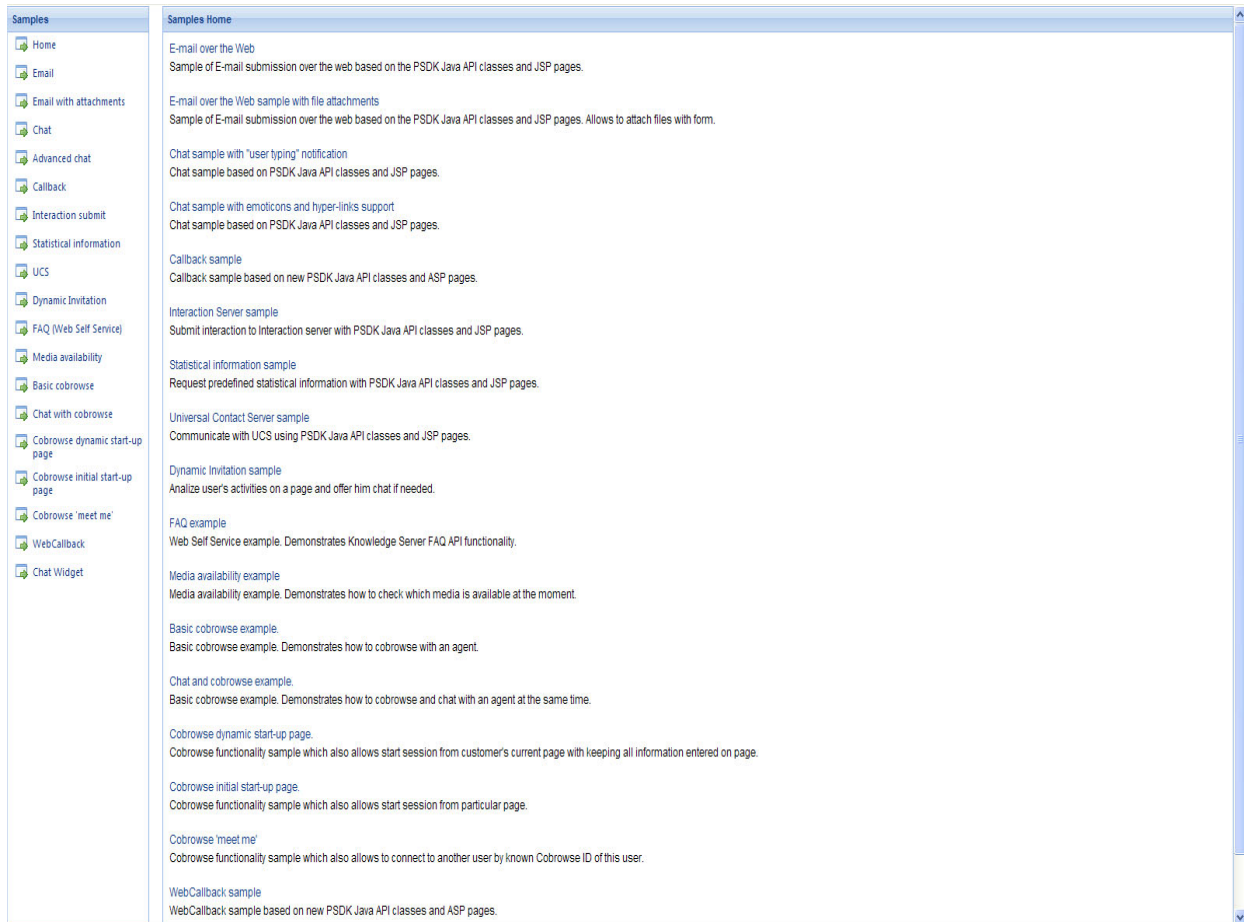


Figure 5: eServices Samples

[Table 6](#) lists these samples and briefly describes them.

Table 6: eServices Samples

Sample	Description
E-mail	Demonstrates how to send an e-mail message.
E-mail with Attachments	Demonstrates how to send an e-mail message that contains an attachment.
Chat	Demonstrates how to initiate a chat session and implement user-typing functionality.

Table 6: eServices Samples (Continued)

Sample	Description
Advanced Chat	Demonstrates how to initiate a chat feature and implements smiles and hyperlink support. This sample also demonstrates how to include a survey in your chat window.
Chat and Cobrowse	Demonstrates how to initiate a chat session and initiate cobrowsing during that session.
Genesys 3rd Party Media (Interaction Submit) ^a	Demonstrates how to use Genesys 3rd Party Media features to create a custom interaction.
Basic Cobrowse	Demonstrates how to initiate a cobrowse session.
Cobrowse with Dynamic Startup Page	Demonstrates how to initiate a cobrowse session with a dynamic startup page.
Cobrowse with Initial Startup Page	Demonstrates how to initiate a cobrowse session with a specified startup page.
Cobrowse with Meet Me	Demonstrates how to initiate a cobrowse “meet me” session.
FAQ (Web Self-Serve)	Demonstrates FAQ functionality.
Statistical Information	Demonstrates how to select predefined statistics and retrieve their current value by using a web form.
Dynamic Invitation	Demonstrates how to analyze the activities of a client on a web page, and enables you to offer to chat with the client.
Media Available	Demonstrates how to check which media are currently available.
Universal Contact Server	Demonstrates how to communicate with Universal Contact Server (UCS) the interaction history of a contact by using a web form.
Web Callback	Demonstrates how to submit a web callback request.
Chat Widget	Demonstrates basic customization of a chat window.

- a. The Genesys 3rd Party Media sample is also referred to as the Interaction Submit Sample, Interaction Server Sample, and Custom Web Form Submit Sample. These all refer to the same sample. The sample is contained in a subdirectory named, `ItxSubmit`.

Test Tools

The Web API installation includes test tools. You can use these tools to verify that the API is working properly and—if you cannot get your application working—to troubleshoot it.

The Web API's Java and .NET implementations each include a load balancing test tool. The Java implementation also provides chat and e-mail test tools.

Installing the Samples

This section identifies where you can find prerequisites and instructions for installing the samples, and tells you how to test their installation.

Note: On some platforms, samples install automatically with product installation, so that you do not need to install them explicitly.

Tools You Need Before Installation

For information on what you will need before installation of the samples, refer to *Genesys Supported Operating Environment Reference Manual*.

Installation Process

For complete installation instructions, see the *eServices 8.0 Deployment Guide*'s chapter on “Model Configuration and Installation On Windows.”

Installation Testing

After installation, use the test tool to verify the installation. Launch a web browser and type the URL of the test tool. The URL should look like the following:

```
http://<web_server_hostname>:<web_server_port>/TestTool80/index.html
```

Configuring for Statistical Information

In order for the “Statistics Information Sample” on [page 57](#) to function, you must:

1. Download the `webapistats.cfg` file. The link to the file can be found on the main page of the Statistical Information Sample.
2. Use Configuration Manager to import the `webapistats.cfg` file into the configuration of all StatServers that handle requests from Web API server.

For a detailed description of this configuration file, refer to Appendix B, “Statistical Information Sample Configuration,” on [page 359](#).

Directory Structure

The following directory structures reflect the standard web client and API server installations.

Java

Installation of the Java implementation produces the following directory structures:

Tomcat

- <tomcat_home>\webapps\WebAPI802\
- <tomcat_home>\webapps\WebAPI802\WebAPISamples802\
- <tomcat_home>\webapps\WebAPI802\TestTool802\

where <tomcat_home> is the directory that contains Tomcat.

.NET

Installation of the .NET implementation produces the following directory structure. Its root, <Target folder>, is the Genesys common-product directory. <WebAPI_Server_App_name> stands for the Configuration Layer’s application name for the Web API Server and Samples application.

- <Target folder>\.NET Web API Server & Samples\<WebAPI_Server_App_name>\
 - Root directory for the WebAPIServer 8.0 web application. It contains some system and support files.
- <Target folder>\.NET Web API Server & Samples\<WebAPI_Server_App_name>\TestTool802\
 - Test Tools web application directory.
- <Target folder>\.NET Web API Server & Samples\<WebAPI_Server_App_name>\SimpleSamples802\
 - Examples web application directory.
- <Target folder>\.NET Web API Server & Samples\<WebAPI_Server_App_name>\SimpleSamples802\Chat\
 - Chat example directory.
- <Target folder>\.NET Web API Server & Samples\<WebAPI_Server_App_name>\SimpleSamples802\ChatAjax\
 - Chat with AJAX example directory.

- <Target folder>\\.NET Web API Server & Samples\<WebAPI_Server_App_name>\SimpleSamples802\Email\
 - E-mail example directory.
- <Target folder>\\.NET Web API Server & Samples\<WebAPI_Server_App_name>\SimpleSamples802\ItxSubmit\
 - Genesys 3rd Party Media example directory.
- <Target folder>\\.NET Web API Server & Samples\<WebAPI_Server_App_name>\SimpleSamples802\Statistics\
 - Stat Server example directory.
- <Target folder>\\.NET Web API Server & Samples\<WebAPI_Server_App_name>\SimpleSamples802\UCS\
 - UCS example directory.
- <Target folder>\\.NET Web API Server & Samples\<WebAPI_Server_App_name>\SimpleSamples802\WebCallback\
 - Web Callback example directory.
- <Target folder>\\.NET Web API Server & Samples\<WebAPI_Server_App_name>\SimpleSamples802\ChatWidget\
 - Chat Widget example directory.
- <Target folder>\\.NET Web API Server & Samples\<WebAPI_Server_App_name>\SimpleSamples802\Survey\
 - Survey example directory.

Sample Files

Each sample demonstrates a particular media feature or a combination of two features. Each sample consists of one or more files.

The samples are categorized by media type:

- Chat
- E-mail
- Genesys 3rd Party Media (Interaction Submit)
- Dynamic Invitation
- Media Availability
- Cobrowse
- FAQ (Web Self-Serve)
- Statistics Information
- UCS
- Survey
- Chat Widget
- Web Callback

Chat Samples

Four samples demonstrate chat-related functionality: Chat, Chat and Cobrowse, Chat Widget, and Advanced Chat. Chat with AJAX is also provided for the .NET implementation.

Basic Chat

The Basic Chat Sample has three to five files, which demonstrate how to create a chat session and implement user-typing functionality:

- `HtmlChatCommand.jsp` (Java) or `ChatCommand.aspx.cs` and `ChatCommand.aspx` (.NET)—Controls the behavior of the chat page, based on user commands
- `HtmlChatFrameSet.jsp` (Java) or `ChatFrameset.htm` (.NET)—Contains the frameset for the other two chat files
- `HtmlChatPanel.jsp` (Java) or `ChatPanel.aspx.cs` and `ChatPanel.aspx` (.NET)—Controls the display in the chat panel

Refer to “Chat Sample” on [page 121](#) for more information.

Advanced Chat Sample

The Advanced Chat Sample has four to five files, which demonstrate how to create a chat session and implement smiles and hyperlink support:

- `HtmlChatCommand.jsp` (Java) or `ChatCommand.aspx.cs` and `ChatCommand.aspx` (.NET)—Controls the behavior of the chat page, based on user commands
- `HtmlChatFrameSet.jsp`—Contains the frameset for the other two chat files.
- `HtmlChatPanel.jsp` (Java) or `ChatPanel.aspx.cs` and `ChatPanel.aspx` (.NET)—Controls the display in the chat panel
- `ChatTranscript.jsp` (Java) or `ChatTranscript.aspx`—Supports popular Internet expressions such as smiling or frowning facial expressions and hyperlinks in an e-mail or chat communication

Refer to “Advanced Chat” on [page 132](#) for more information. This sample also demonstrates how to include a survey in your chat window, see “Survey” on [page 52](#) for information about this functionality.

Chat and Cobrowse

The Chat and Cobrowse Sample demonstrates how to initiate a chat session and initiate cobrowsing during that session. Its file base is almost identical to

those of the two samples on which it is based: Chat, and Cobrowse. For more information on this sample, see “Chat and Cobrowse Sample” on [page 203](#).

Chat with AJAX

The Chat with AJAX Sample (provided only for .NET in this release) has four files, which demonstrate how to create a chat session:

- `ChatCommand.aspx.cs` and `ChatCommand.aspx`—Control the behavior of the chat page, based on user commands
- `ChatPanel.aspx.cs` and `ChatPanel.aspx`—Control the display in the chat panel

Refer to “Chat with AJAX Sample” on [page 251](#) for more information.

Chat Widget

The Chat Widget Sample has five to eight files, which demonstrate how to customize a chat window:

- `HtmlChatCommand.jsp` (Java) or `ChatCommand.aspx.cs` and `ChatCommand.aspx` (.NET)—Controls the behavior of the chat page, based on user commands
- `HtmlChatFrameSet.jsp`—Contains the frameset for the other two chat files
- `HtmlChatPanel.jsp` (Java) or `ChatPanel.aspx.cs` and `ChatPanel.aspx` (.NET)—Controls the display in the chat panel
- `ChatTranscript.jsp` (Java) or `ChatTranscript.aspx` and `ChatTranscript.aspx.cs` (.NET)—Presents the transcript, including smiling or frowning facial expressions and hyperlinks in a chat communication
- `ChatWidget.html`—Contains the main form where you can customize the chat window
- `Available.aspx`—Emulates the verification that an agent is available to chat with the customer. Currently, this file always replies that an agent is always available, but it can be modified to work according to any set of business rules.

Refer to “Chat Widget Sample” on [page 134](#) for more information.

Survey

The Advanced Chat Sample demonstrates survey functionality, and consists of either one or two files:

- Either `Survey.jsp` (Java), or `Survey.aspx.cs` and `Survey.aspx` (.NET)—Contain(s) code snippets that use the survey portion of the API to initialize a survey

Refer to “Survey Sample” on [page 138](#) for more information on this sample.

E-Mail Samples

Two samples demonstrate e-mail functionality: E-mail and E-mail with Attachments. In this release, the .NET implementation combines the features of both the E-mail and the E-mail with Attachments samples in a single E-mail Sample.

E-Mail

The E-mail Sample demonstrates e-mail functionality, and consists of either one or two files:

- Either `Email.jsp` (Java), or `Email.aspx.cs` and `Email.aspx` (.NET)—Contain(s) code snippets that use the e-mail portion of the Web API to send e-mail messages

Refer to “E-Mail Sample” on [page 108](#) for more information on this sample.

E-Mail with Attachments

The E-mail with Attachments Sample demonstrates how to create an e-mail message that includes an attachment, and consists of only one file:

- `Email.jsp`—Contains Java code snippets that use the e-mail portion of the Web API to send e-mail messages that include attachments

Refer to “E-Mail with Attachments Sample” on [page 113](#) for more information on this sample. The .NET “E-Mail Sample” on [page 269](#) also contains attachments functionality.

Web Callback Sample

One sample demonstrates web callback functionality. It consists of either one or two files:

- Either `WebCallback.jsp` (Java) or `WebCallback.aspx.cs` and `WebCallback.aspx` (.NET)—Contain(s) code snippets that use the callback portion of the Web API to submit a web callback request

Refer to “Web Callback Sample” on [page 144](#) for more information on this sample.

Genesys 3rd Party Media Sample

One sample demonstrates Genesys 3rd Party Media functionality. This sample consists of either one or two files:

- Either `ItxSubmit.jsp` (Java) or `ItxSubmit.aspx.cs` and `ItxSubmit.aspx` (.NET)—Contain(s) code snippets that use the Genesys 3rd Party Media portion of the Web API to submit a custom interaction

Refer to “Interaction Submit (Genesys 3rd Party Media) Sample” on [page 151](#) for more information on this sample.

Dynamic Invitation Sample

One sample demonstrates Dynamic Invitations functionality. It consists of three files:

- `ajax.js`—Contains the AJAX functionality that is required for sending HTTP requests via XMLHttpRequest objects
- `Available.jsp` (Java) or `Available.aspx` (.NET)—Receives the AJAX requests and checks the availability of the required media before it displays the pop-up screen
- `Register.jsp` (Java) or `Register.aspx` (.NET)—Contains the code that is required to display the pop-up screen, process the AJAX requests, and control the user activities on a form. Based on analyses of these activities an internal JavaScript decides when to display a pop-up window and when to offer chat, e-mail, or Web Self-Serve (FAQ) options to the customer.

Refer to “Dynamic Invitation Sample” on [page 181](#) for more information on this sample.

Media Availability Sample

One sample demonstrates Media Availability functionality. It consists of only one file:

- `MediaAvailability.jsp`—Demonstrates how to check which media are available at the moment

Refer to “Media Availability Sample” on [page 193](#) for more information on this sample.

Cobrowse Samples

Five samples demonstrate web collaboration-related functionality: Basic Cobrowse, Chat and Cobrowse, Cobrowse with Dynamic Startup Page, Cobrowse with Initial Startup Page, and Cobrowse with Meet Me.

Basic Cobrowse

One sample demonstrates Basic Cobrowse functionality. It is based on the Cobrowsing Server API and consists of nine files:

- `blank.html`—Initializes empty frames that are used by the API. This is a default page
- `hbapi.html`—Supports the client-side API and the cobrowse applet
- `hbmessage_to_var.html`—A messaging file that provides messages from the cobrowse frame to the web-application frame
- `hbmessage_to_var.js`—A messaging file that provides messages from the cobrowse frame to the web-application frame

- `hbmessaging.js`—A JavaScript file that contains the API of Cobrowsing Server
- `hbmessagingform.html`—Increases security by sending Agent login information as an HTTP POST request, instead of as a GET request
- `qstring.js`—A JavaScript file that contains a utility class
- `CoBrowse.htm`—The main frameset of the cobrowse sample
- `CoBrowseEventHandler.jsp` (Java) or `CoBrowseEventHandler.aspx` and `CoBrowseEventHandler.aspx.cs` (.NET)—The cobrowse event-receiver frame for the “meet me” sample

Refer to “Basic Cobrowse Sample” on [page 197](#) for more information on this sample.

Note: The `qstring.js`, `hbmessage_to_var.html`, and `hbmessage_to_var.js` files must all reside in the same directory.

Chat and Cobrowse

The Chat and Cobrowse Sample demonstrates how to create a chat session and use web-collaboration features. It consists of 12 files:

- `blank.html`—Initializes empty frames that are used by the API. This is a default page.
- `ChatAndCoBrowse.htm`—The main frameset of the sample
- `ChatTranscript.jsp` (Java) or `ChatTranscript.aspx` (.NET)—Supports popular Internet expressions such as smiling or frowning facial expressions and hyperlinks in an e-mail or chat communication
- `hbapi.html`—Supports the client-side API and the cobrowse applet
- `hbmessage_to_var.html`—A messaging file that provides messages from the cobrowse frame to the web-application frame
- `hbmessage_to_var.js`—A messaging file that provides messages from the cobrowse frame to the web-application frame
- `hbmessaging.js`—A JavaScript file that contains the API of Cobrowsing Server
- `hbmessagingform.html`—Increases security by sending Agent login information as an HTTP POST request, instead of as a GET request
- `qstring.js`—A JavaScript file that contains a utility class
- `HtmlChatCommand.jsp` (Java) or `ChatCommand.aspx` and `ChatCommand.aspx.cs` (.NET)—Controls the behavior of the chat page, based on user commands
- `HtmlChatFrameSet.jsp`—Contains the frameset for the other two chat files
- `HtmlChatPanel.jsp` (Java) or `ChatPanel.aspx` and `ChatPanel.aspx.cs` (.NET)—Controls the display on the chat panel and handles cobrowse functionality

Note: The `qstring.js`, `hbmessage_to_var.html`, and `hbmessage_to_var.js` files must all reside in the same directory.

Refer to “Chat and Cobrowse Sample” on [page 203](#) for more information on this sample.

Cobrowse with Dynamic Startup Page

One sample demonstrates Cobrowse with Dynamic Startup Page functionality. Its core logic resides in these file:

- `ExampleOfDynamicStartupPage.jsp` (Java) or `ExampleOfDynamicStartupPage.aspx` and `ExampleOfDynamicStartupPage.aspx.cs` (.NET)

The sample’s remaining files should not be manipulated in any way:

- `responseLive.js`
- `responseLiveLauncher.html`
- `responseLiveScriptletLauncher.html`
- `responseLiveStartApp.html`

Refer to “Cobrowse with Dynamic Startup Page Sample” on [page 218](#) for more information on this sample.

Cobrowse with Initial Startup Page

One sample demonstrates Cobrowse with Initial Startup Page functionality. It consists of nine files:

- `blank.html`—Initializes empty frames that are used by the API. This is a default page.
- `CoBrowseEventHandler.jsp`—The cobrowse event-receiver frame.
- `hbapi.html`—Supports the client-side API and the cobrowse applet.
- `hbmessage_to_var.html`—A messaging file that provides messages from the cobrowse frame to the web-application frame.
- `hbmessage_to_var.js`—A messaging file that provides messages from the cobrowse frame to the web-application frame.
- `hbmessaging.js`—A JavaScript file that contains the API of Cobrowsing Server.
- `hbmessagingform.html`—Increases security by sending Agent login information as an HTTP POST request, instead of as a GET request.
- `InitialStartupPageExample.html` (Java) or `InitialStartupPageExample.aspx` and `InitialStartupPageExample.aspx.cs` (.NET)—Start a cobrowser session from this page.
- `qstring.js`—A JavaScript file that contains a utility class.

Refer to “Cobrowse with Initial Startup Page” on [page 214](#) for more information on this sample.

Cobrowse with Meet Me

One sample demonstrates Cobrowse with Meet Me functionality. It consists of nine files:

- `blank.html`—Initializes empty frames that are used by the API. This is a default page.
- `CoBrowse.htm`—The main frameset of the cobrowse sample.
- `CoBrowseEventHandler.jsp` (Java) or `CoBrowseEventHandler.aspx` and `CoBrowseEventHandler.aspx.cs` (.NET)—The cobrowse event-receiver frame for the “meet me” sample.
- `hbapi.html`—Supports the client-side API and the cobrowse applet.
- `hbmessage_to_var.html`—A messaging file that provides messages from the cobrowse frame to the web-application frame.
- `hbmessage_to_var.js`—A messaging file that provides messages from the cobrowse frame to the web-application frame.
- `hbmessaging.js`—A JavaScript file that contains the API of Cobrowsing Server.
- `hbmessagingform.html`—Increases security by sending Agent login information as an HTTP POST request, instead of as a GET request.
- `qstring.js`—A JavaScript file that contains a utility class.

Refer to “Cobrowse with Meet Me” on [page 211](#) for more information on this sample.

FAQ Sample for Java

One sample demonstrates FAQ (self-serve Knowledge Base) functionality. It consists of only one file:

- `FAQ.jsp`—Dynamic FAQ sample main page

Refer to “FAQ (Web Self-Serve) Sample” on [page 187](#) for more information on this sample.

Statistics Information Sample

One sample demonstrates statistics functionality. It consists of the following files:

- `StatInfo.jsp` (Java) or `StatInfo.aspx.cs` and `StatInfo.aspx` (.NET)—Contain(s) code snippets that retrieve statistics from Stat Server about interactions

Refer to “Statistical Information Sample” on [page 160](#) for more information on this sample.

Universal Contact Server Sample

The Universal Contact Server sample consists of the following files, which demonstrate e-mail history functionality:

- `Action.jsp` (Java) or `Action.aspx.cs` and `Action.aspx` (.NET)—Processes user command, such as; Mark as read, Print, and Print thread
- `UCS.jsp` (Java) or `UCS.aspx.cs` and `UCS.aspx` (.NET)—Controls most of the page presentation to the user

Refer to “Universal Contact Server Sample” on [page 166](#) for more information.

Test Tool Files

Java

The Java implementation provides the following test tools:

- `blank.html`—Initializes empty frames that are used by the API. This is a default page.
- `ChatSelfTest.jsp`—Validates eServices chat web API.
- `EmailSelfTest.jsp`—Validates eServices e-mail web API.
- `index.html`—The primary entry point to the Test Tools.
- `LBTestPage.jsp`—Validates system load balancing installation.
- `StatSelfTest.jsp`—Validates web statistics API.
- `UCSSelfTest.jsp`—Validates UCS web API.
- `TestToolMenu.jsp`—Main menu page.

.NET

The .NET implementation provides the following test tools:

- `diagnostic.xsl`—Stylesheet file for diagnostic routines.
- `LBTest.aspx.cs`—Partial class for the `LBTest.aspx` page. It validates the system load balancing installation.
- `LBTest.aspx`—Displays the `LBTest.aspx.cs` file.
- `EmailSelfTest.aspx.cs`—Partial class for `EmailSelfTest.aspx` page. It validates the eServices e-mail .NET API.
- `EmailSelfTest.aspx`—Displays the `EmailSelfTest.aspx.cs` file.
- `ChatSelfTest.aspx.cs`—Partial class for the `ChatSelfTest.aspx` page. It validates the eServices chat.NET API.
- `ChatSelfTest.aspx`—Displays the `ChatSelfTest.aspx.cs` file.

- `StatSelfTest.aspx.cs`—Partial class for the `StatSelfTest.aspx` page. It validates the eServices stat server .NET API.
- `StatSelfTest.aspx`—Displays the `StatSelfTest.aspx.cs` file.
- `UCSSelfTest.aspx.cs`—Partial class for the `UCSSelfTest.aspx` page. It validates the eServices Universal Contact Server .NET API.
- `UCSSelfTest.aspx`—Displays the `UCSSelfTest.aspx.cs` file.



Chapter

3

Understanding and Using the E-Mail Service

eServices's e-mail service is powerful but fairly complicated. It involves many servers, and is tied to routing and workflow strategies, statistics, classification, and knowledge-management tasks. Because deploying an e-mail service involves many development groups and levels, this chapter aims to present an overview of the e-mail service, and to explain the service from a web-application perspective.

This chapter explains how you can use this service in a web application, without knowing what happens to the e-mail after submission. It does not attempt to address any strategies, intelligent routing, or the classification mechanism. It contains the following sections:

- [Overview, page 61](#)
- [Life Cycle of an E-Mail Session, page 62](#)
- [Event Flow of an E-Mail Session, page 62](#)

To fully understand the e-mail service, consult the *eServices 8.0 Deployment Guide*, *eServices 8.0 User's Guide*, *Universal Routing 8.0 Business Process User's Guide*, *Universal Routing 8.0 Reference Manual*, and *Framework 8.0 Stat Server User's Guide*.

Overview

Figure 6 on [page 62](#) shows the architecture and components involved in an e-mail service. The *eServices 8.0 Deployment Guide* explains the behavior of E-mail Server in greater detail.

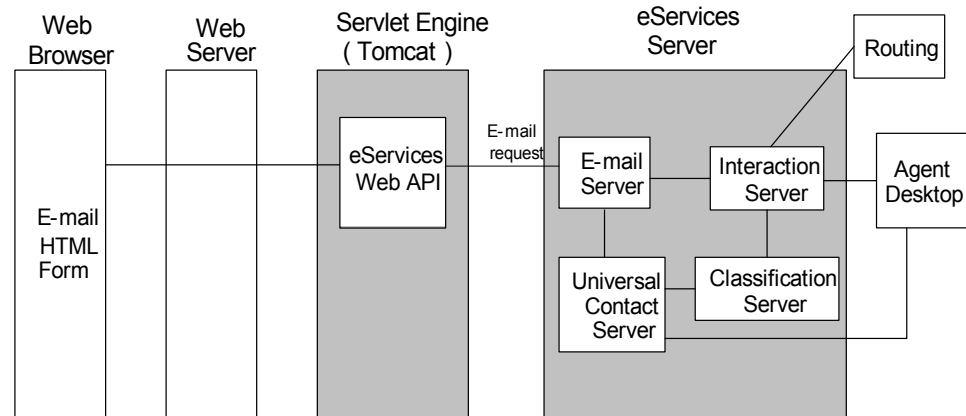


Figure 6: Web-Based E-Mail Service Architecture

Life Cycle of an E-Mail Session

The life cycle of an e-mail session is quite complicated. Figure 7 depicts this life cycle from the perspective of the eServices E-mail Sample. The initial state is the empty state. The only action available here is to connect to E-mail Server. After your application logs in, the user can either submit the e-mail form immediately, or ask to see statistics. In either case, the action moves the form from the logged in state to the in session state. However, after submission, the web client immediately executes the close action and changes the state to empty again.

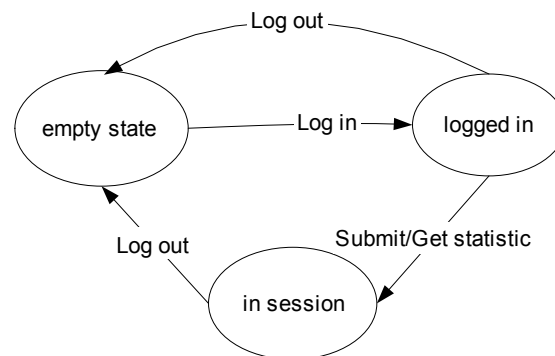


Figure 7: State Diagram of an E-Mail Session

Event Flow of an E-Mail Session

This section presents the event flow of an e-mail session, from a web-client perspective. Figure 8 on page 63 shows the event flow from the Web sample to E-mail Server. The rightmost box represents the workflow-control

components, such as knowledge management (classification), routing, and strategies, if any, as a single entity.

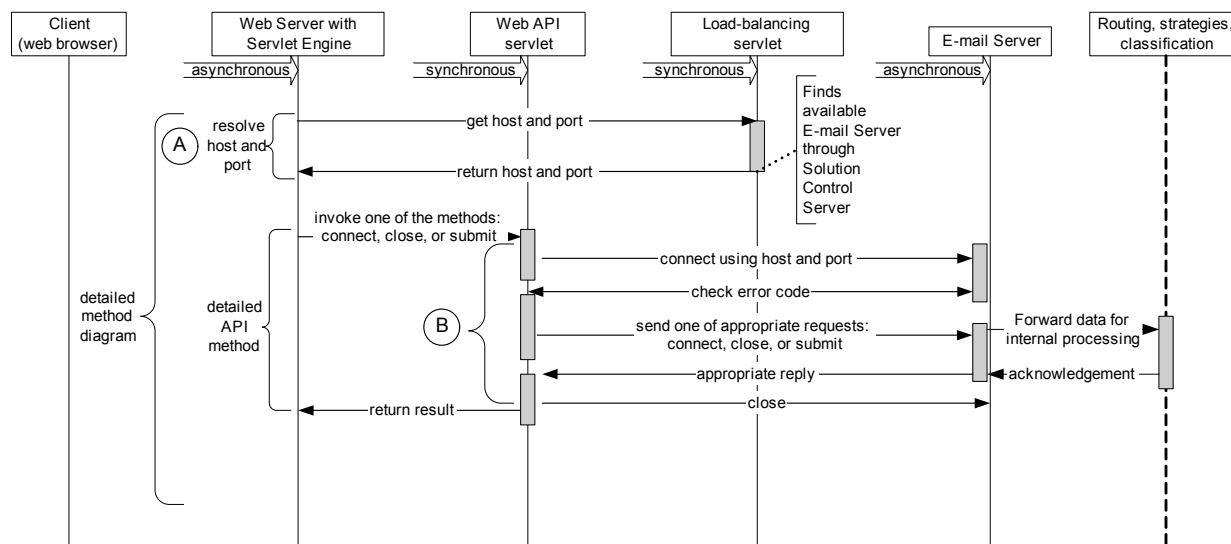


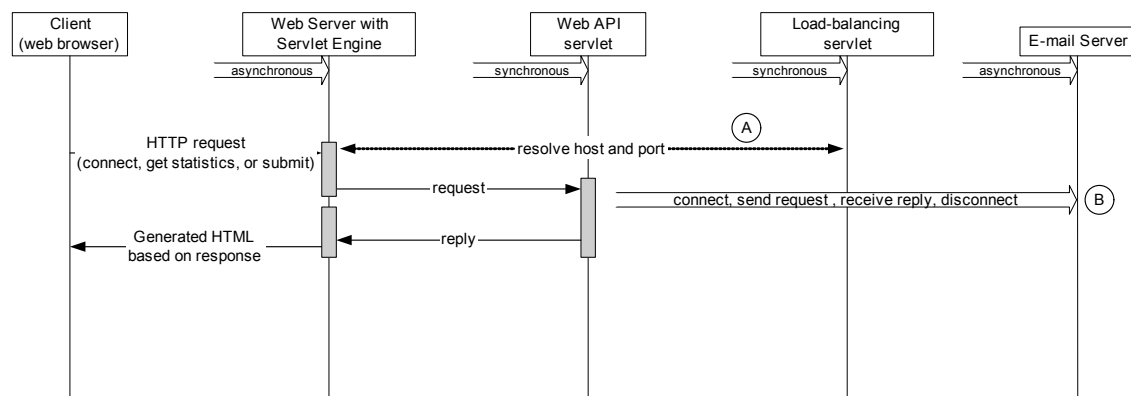
Figure 8: Event Flow Between the Web API Servlet and E-Mail Server

Note: Figure 8 shows a detailed view of the event flow between the Web API servlet and E-mail Server. It identifies certain sections of the event flow with the labels A and B. Figure 9 on [page 64](#) abstracts those sections of the diagram by referencing these labels.

Your web application gets a host name and port number from load balancing, and uses them to create an `_irs_direct` object that is connected to the specified host and port. The Web API's customized server page (JSP or ASPX) then forwards the request to E-mail Server. E-mail Server then forwards the request data to Interaction Server, which in turn repackages it and forwards it for further processing. At this point, the `_irs_direct` object must close the connection. Strictly speaking, the web-based e-mail architecture ends at E-mail Server. Any subsequent activity is not considered to be part of the e-mail architecture, but rather part of the eServices architecture.

Figure 9 on [page 64](#) depicts the event flow of an HTTP request for connection, statistic retrieval, or submission.

For more information on how eServices processes an e-mail, refer to the *eServices 8.0 Deployment Guide*.

**Figure 9: Event Flow for a Connection, Statistic Retrieval, or Submission Request**



Chapter

4

Understanding and Using the Flex Chat Service

This chapter details the chat service in eServices, and the Flex Chat API concept. It contains the following topics:

- [Overview, page 65](#)
- [Life Cycle of a Chat Session, page 66](#)
- [Event Flow of a Chat Session, page 67](#)
- [Transcripts, page 69](#)

Overview

This section illustrates the concepts and architecture for the chat service.

[Figure 10](#) depicts the components involved in a chat service. This chapter only discusses the browser, web server, Web API, and Chat Server. The *eServices 8.0 Deployment Guide* discusses Chat Server in more detail.

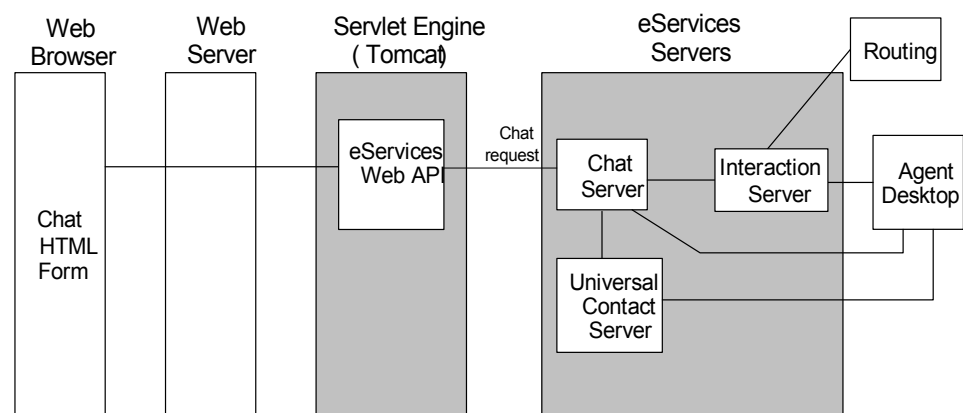


Figure 10: Architecture of the Flex Chat Service

Life Cycle of a Chat Session

To successfully deploy a chat service onto your web application, you must first understand the life cycle of a chat session. [Figure 11](#) shows the life cycle as a state diagram. The start state is the empty state. The only action available here is to log in to Chat Server as a chat user. Note that your application can log out of Chat Server without attempting to create or join a session (logged in state). However, this is not the normal flow of a chat session. It usually indicates that an error or exception has occurred.

Warning! Your client application's code must disable the Stop chat button until the application receives a reply from Chat Server to the browser's Connect request. This avoids prematurely ending a chat (by sending a `requestLogout` to Chat Server). Such premature logouts leave behind pending interactions that Genesys Desktop will be unable to delete.

After your customer is logged in, your application must call the `join()` method. If `join()` is invoked with a null `session_id` parameter, a new session is created. If it is invoked with an existing session ID, your customer is reconnected with that session.

Note: The eServices Web API samples show how to use the `join()` method to create a new session. A customer logged into Chat Server can participate in only one session. To connect or create another chat session, the customer needs to perform another login, which will produce a separate `user_id`.

These samples do not show how to connect to an existing session. For that information, consult the *Platform SDK 8.0 Java (or .NET) API Reference*.

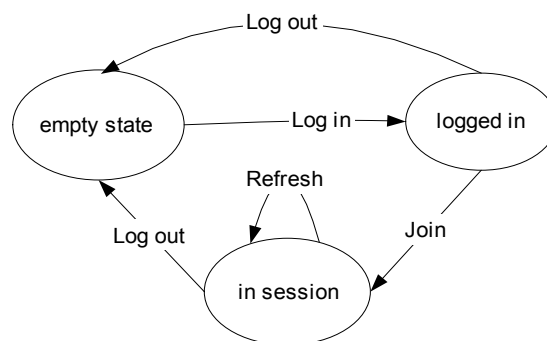


Figure 11: State Diagram of a Chat Session

Once a chat session starts, your application must periodically send a `refresh` request to keep the session alive. The refresh frequency is dependent on your application. However, you can use the `flex-disconnect-timeout` configuration option in Chat Server to specify the maximum timeout between refresh requests. Make sure that the interval in which your application calls the `refresh()` method falls within the `flex-disconnect-timeout` value. Your application stays in the `in session` state until it logs out of Chat Server.

Note: You can view the life cycle of a chat session from two perspectives: that of the Chat Server and that of a chat participant. From Chat Server's point of view, the session begins when the first participant joins, and continues until the last participant disconnects from the session. From a user's point of view, the chat session is over when that user disconnects. The user receives the transcript up to the point where he or she stopped chatting. However, Chat Server is still servicing the other users, and their transcripts will be different. See the section "Transcripts" on [page 69](#) for more information.

Event Flow of a Chat Session

This section details the event flow of a typical chat session (see Figure 12 on [page 68](#)).

In the first request in a chat session, the API servlet must request the alias for a Chat Server (flow A in Figure 12 on [page 68](#)). All subsequent requests in the chat session must use this alias to resolve the host and port information from the load balancing servlet before invoking any method of the Chat API (flow B).

Note: The same Chat Server is used for the whole chat session (see "Life Cycle of a Chat Session" on [page 66](#)).

When your application makes a `login`, `logout`, `join`, or `refresh` request, the API servlet must connect to the Chat Server before it can forward your application request (flow C). Chat Server sends a reply, with a chat transcript if applicable. Then the servlet disconnects from Chat Server. Note that the Web API connects to, and disconnects from, Chat Server for each `login`, `logout`, `join`, or `refresh` method call.

Note: [Figure 12](#) details the event flow between the Web API servlet and Chat Server. It identifies certain sections of the event flow using the labels A, B, and C. [Figure 13](#) on [page 68](#) and [Figure 14](#) on [page 69](#) abstract those sections of the diagram by referencing these labels.

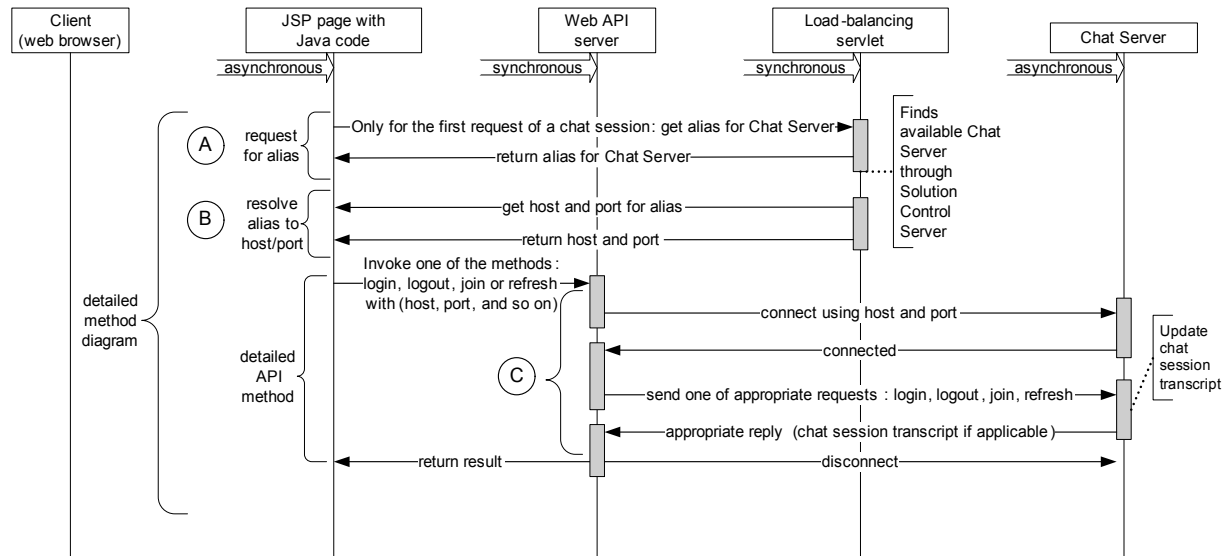


Figure 12: Request/Event Flow Between eServices Components for Chat Service

Figure 13 shows the event flow of a connect request. Your web application sends an HTTP request containing a `cmd = connect` parameter to the servlet (through the Web API server). The double-ended arrow between the server page (labeled JSP) and the load balancing servlet is an abstraction of the event flows labeled “request for alias” and “resolve alias to host/port” in Figure 12.

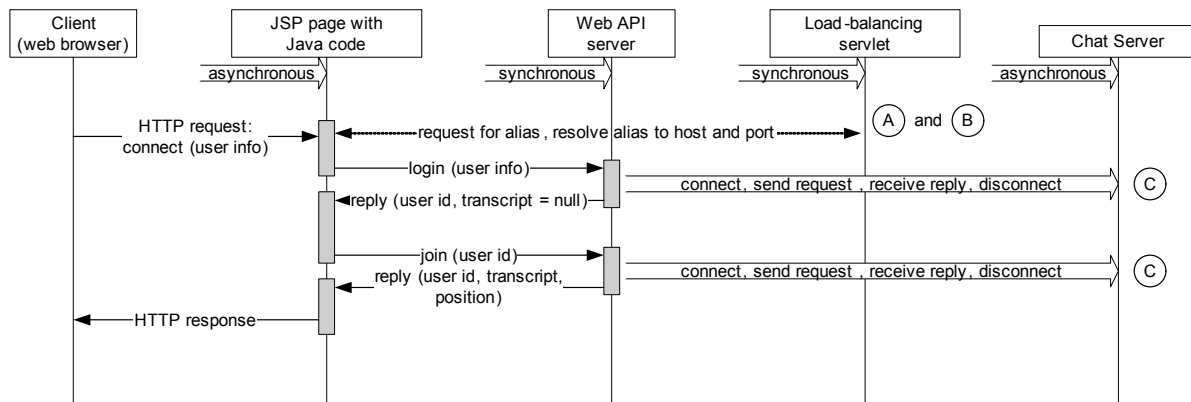


Figure 13: Event Flow for Connect Request

Figure 14 on page 69 shows the event flow for an HTTP request with a `cmd = send` or `cmd = disconnect` parameter. As with the `cmd = disconnect` parameter, the Web API server must connect to Chat Server, forward the request, and then disconnect from Chat Server. Note that a web client must send the chat alias for each request.

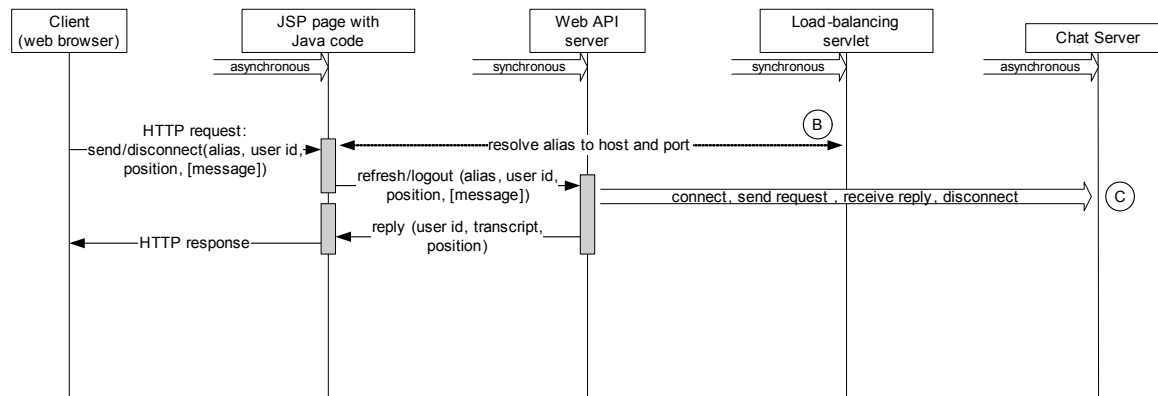


Figure 14: Event Flow for Send or Disconnect Request

Transcripts

A transcript is a set of ordered items from a chat session. Each item represents an event in that session. Each reply from Chat Server in the Flex protocol, except a reply for a connect request, contains such a transcript. (The reply for a connect request contains no transcript because a chat session has not yet been established.)

A transcript contains only recent events in the chat session (those generated since the last refresh request). It can even be empty (that is, it can contain no items) if no updates were made to the chat session since the previous refresh request.

Transcript Description

A Flex transcript contains a header and a list of events.

The header contains the following data:

1. `session_id` (the same as `interaction id`)—required for statistics inquiries.
2. `start_at`—timestamp showing when chat session was created.
3. `last_position`—the ordering number of the last transcript item. This value is needed for subsequent refresh requests.

Each transcript item contains the following data:

1. `event_type`, which could be one of three types:
 - a. `CONNECT` specifies that a party (either a web client or an agent) joined a chat session. A transcript always starts with an item of this type for the client that requested the session creation.
 - b. `MESSAGE` specifies that a message was sent by someone already participating in the chat session. (Therefore, a `CONNECT` item was already specified for this party.)
 - c. `ABANDON` specifies that a participant left the chat session.

2. `time_offset` is the number of seconds that elapsed from the time of session creation.
3. `user_nick` represents user nickname information.
4. `user_type`, which is either `AGENT` or `CLIENT`. According to the current implementation of the web samples, only one party of type `CLIENT` is possible in one chat session. However, this restriction may be relaxed in future releases.
5. `party_id` contains a unique identifier of a party in a chat session. However, that identifier is not unique across different chat sessions.
6. `msg_type` is not currently used. `msg_type` is reserved for future specification of the message content. However, this field could contain text, such as `TEXT`.
7. `event_body` contains the message itself. Each transcript item (not just those of type `MESSAGE`) may contain an event body.

API Representation

The `_chat_transcript` class in the Web API is a vector of transcript items (objects). It contains members as described in “[Transcript Description](#),” above. The `transcript()` method of the `_chat_direct` class returns a transcript in the case of successful reply.

Note: No transcript can be requested immediately after a call to the `login` method.



Chapter

5

Understanding and Using the Genesys 3rd Party Media Service

This chapter discusses the Genesys 3rd Party Media service, in the following sections:

- [Overview, page 71](#)
- [Architecture, page 72](#)
- [Event Flow of a Request, page 72](#)

Overview

Genesys 3rd Party Media is designed to enable contact centers to manage all communication channels in the same way, with the same pool of agents, and with consistent reporting.

In the broadest sense, Genesys 3rd Party Media lets you define your own media types, such as fax or customer-relationship management (CRM) cases. You can then pass these media types to and from Interaction Server in much the same way that you would use eServices's built-in chat or e-mail types.

The Web API implementation of Genesys 3rd Party Media provides a more focused set of capabilities. In particular, you can use the Web API to write customized server pages that gather user input on a web form. This input is sent to the Web API Server, which creates a new interaction and passes it on to Interaction Server for further processing by the Genesys platform.

Architecture

eServices's implementation of Genesys 3rd Party Media relies on Interaction Server, as shown in [Figure 15](#). This simplified diagram indicates that an interaction can be initiated from a web browser, created in a web server, and sent to Interaction Server. From there, it can optionally be passed to another component—such as the Genesys Classification Server or a custom application—for further processing.

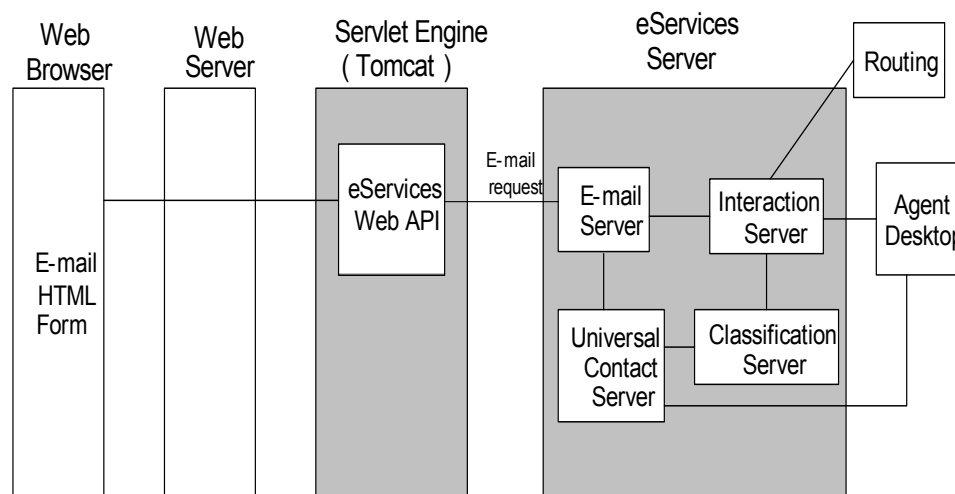


Figure 15: Web-Based Genesys 3rd Party Media Service Architecture

Event Flow of a Request

Figure 16 on [page 73](#) shows the event flow from the Web sample, which is running on a browser.

The Web sample's advanced server page submits the web form to the Web API Server. The Web API Server creates a new interaction based on the user input, and submits it to Interaction Server.

When the Interaction Server receives the interaction, it may optionally pass the interaction to a custom application for further processing. Otherwise, the interaction is processed by the Genesys platform in the same way that other interactions are processed.

When Interaction Server receives an interaction from the Web API Server, it sends an acknowledgement. When the Web API Server receives that acknowledgement, it in turn sends an acknowledgement to the browser.

Other types of interaction-related activity, such as an update or cancellation, have the same structure as is shown in Figure 16 on [page 73](#).

Note: You must ensure that the interaction ID is unique in order for the Genesys platform to process it correctly.

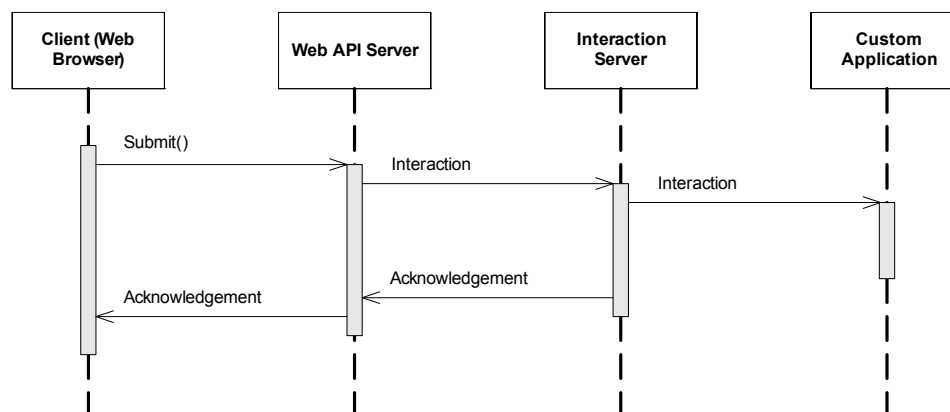


Figure 16: Event Flow Between the Interaction Server and Other Components



Chapter

6

Understanding and Using the Web Collaboration Service

This chapter explains the concept of web collaboration, and provides a high-level view of Cobrowsing Server's architecture. This server adds cobrowse capabilities to eServices web-based clients.

Note: Cobrowse samples are provided only for Java in this release.

The chapter contains the following sections:

- [What is Cobrowsing?, page 75](#)
- [Architecture, page 76](#)
- [Web Collaboration Process, page 76](#)
- [Integrating Cobrowsing into Your Application, page 78](#)

What is Cobrowsing?

Cobrowsing Server is a web-based application that enables agents and customers to cobrowse web pages: that is, to view the same web page together, with one party's actions on the page being instantly propagated to the other party's browser. For example, if you and another person are cobrowsing the Yahoo! home page, when you click the News link, your cobrowsing partner also sees the Yahoo! News page.

The actions that the participants in a cobrowsing session can perform together are:

- Navigating web sites.
- Conducting online transactions.

- Filling out web forms.
- Interacting with web-based software applications.
- Downloading files, playing audio, or watching video streams.

Architecture

Cobrowsing Server acts as a proxy between the clients and the site navigated. Cobrowsing Server sits between the clients and the servers at the website.

[Figure 17](#) shows Cobrowsing Server's high-level architecture.

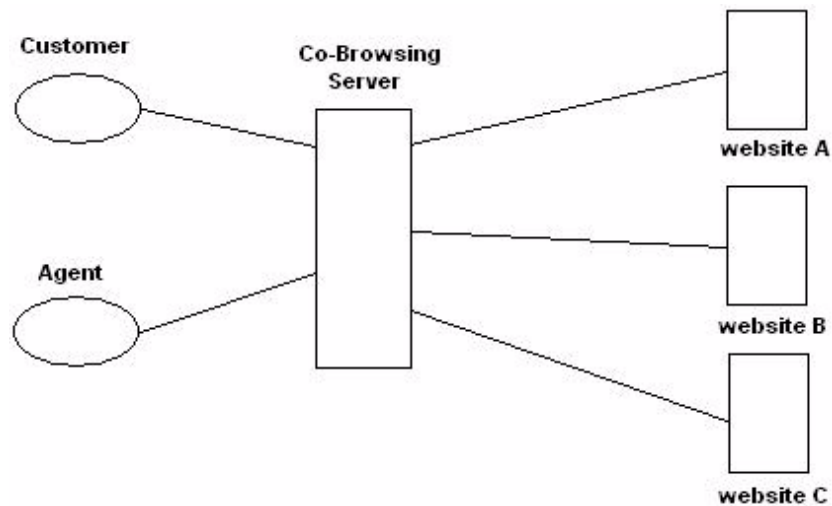


Figure 17: Cobrowsing Server Proxy Architecture

Web Collaboration Process

Figure 18 on [page 77](#) shows the typical sequence of steps in the web collaboration process between an agent and a customer.

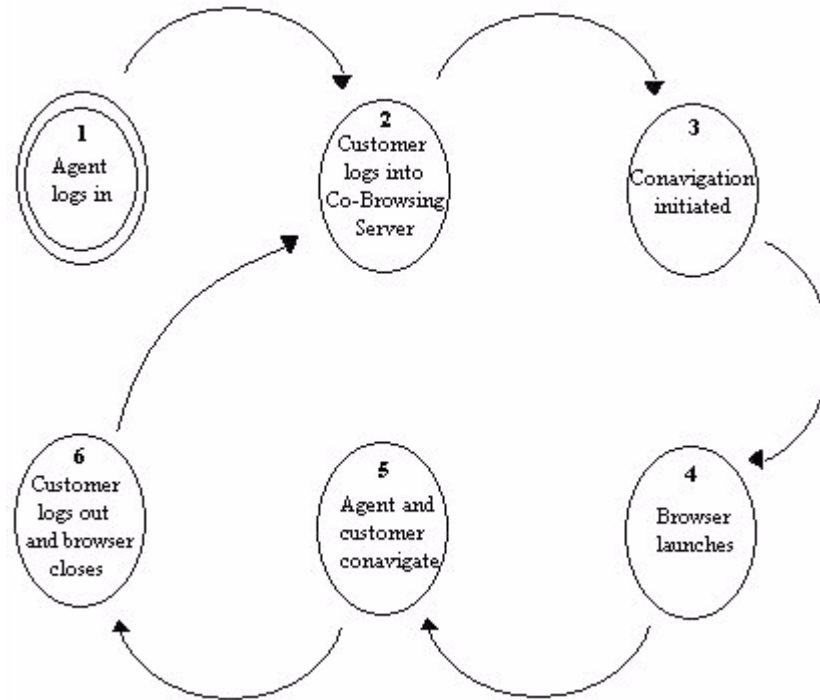


Figure 18: Web Collaboration Process

1. Agent logs in to Interaction Server through an agent desktop application.
2. Customer logs in to Cobrowsing Server. While communicating with the agent, your application uses the Cobrowsing Server API to automatically log the customer in to Cobrowsing Server and load the cobrowse applet. The applet is loaded in the background, in preparation for the cobrowse session.
3. Customer initiates a cobrowse session: The customer application sends an `initiate co-browse` command, and the customer's cobrowse ID, to the agent application.
4. The agent browser launches when the agent application invokes the Cobrowsing Server API method `HBConnectTo` (specifying the customer's cobrowse ID). This connects the agent and customer, and launches shared browsers on both computers.
5. Cobrowsing commences. The customer and agent cobrowse.
6. Customer logs out of Cobrowsing Server. The cobrowse client closes the customer browser and logs the customer out. The agent is free to start another session with a new customer.

Note: Cobrowsing Server sends a confirmation message when the customer is logged out. The customer application or service can trap these events by using the JavaScript calls provided in the Cobrowsing Server API.

Integrating Cobrowsing into Your Application

In order to integrate cobrowse into your client application, you must modify your application to include some cobrowse support files. These files contain API calls that allow your client application to log customers in, initiate web collaboration sessions, navigate to websites, and log customers out.

This section lists the files that your client must include, and describes how to use the APIs to perform mandatory operations. It also provides example code snippets that suggest ways on how to create initial and dynamic startup pages, how to disable submit buttons, and how to take account of other features and limitations for creating a client application.

Integration Support Files

- `hbmessaging.js`—JavaScript file containing Cobrowsing Server's API.
- `qstring.js`—JavaScript file containing a utility class.
- `blank.html`—default blank page for initializing empty frames used by the API.
- `hbapi.html`—supports the client-side API and the cobrowse applet.
- `hbmessagingform.html`—increases security by sending Agent login information as a HTTP POST request instead of a GET request.
- `hbmessage_to_var.html`—A messaging file that provides messages from the cobrowse frame to the web application frame.
- `hbmessage_to_var.js`—A messaging file that provides messages from the cobrowse frame to the web application frame.

Note: `qstring.js`, `hbmessage_to_var.html`, and `hbmessage_to_var.js` must all reside in the same directory.

Example Client Files

The following web HTML files come with the Cobrowsing Server installation. The default installation puts them under the `<Co-browse server home>/clientapi/` directory.

- `topLevelframeset.html`—a frameset that holds the frames for cobrowse messaging files.
- `varExampleForm.html`—a simple example client that captures and sends events to Cobrowsing Server.
- `examples.html`—a sample implementation of client callbacks provided by the cobrowse applet.

Design, Deployment, and Configuration Details

For cobrowsing application design, deployment, and configuration guidelines, see the KANA Response Live documentation listed in “Related Documentation Resources” on [page 363](#). (Genesys licenses certain co-browsing components from KANA Software, Inc.)

Creating a Cobrowse Session in a Separate Browser Window

The Cobrowsing Server client API supports two different ways of creating cobrowse sessions:

- Within the embedded frame.
- In a new browser window. This is the only alternative supported on Windows XP (SP2 and up), Windows 2003, or Windows Vista. An example of this functionality is the Cobrowse sample that is described in “Cobrowse Samples” on [page 54](#).

You cannot use the `HBInitEmbeddedFrame` function when you have a cobrowse session in a separate window. A new co-browse session window appears immediately after invocation of the `HBConavigateLink` function. If the client invokes `HBExitSession`, the window disappears.

Sample Code Snippet

```
function CoBrowse()
{
    var HBApiWindow = parent.hbapi;
    var strUrl = new String(window.location.href);
    var strProcessorUrl = strUrl.substring(0, strUrl.lastIndexOf("/")+
        hbmessage_to_var.html");
    var CobrowseHostName = "cobrowse.genesyslab.com";
    var AttachedData = "acctSpecificData";
    var ConavigationiChannelID = "Default";

    HBApiWindow.HBInitializeAPI("EventHandlerFrame", CobrowseHostName, strProcessorUrl);
    HBApiWindow.HBLoginGuest(ConavigationiChannelID, "", "", false, AttachedData);
}
//HB API Event Handler
function HBSessionStarted()
{
    var StartPage = "http://www.genesyslab.com";
    HBApiWindow.HBConavigateLink(StartPage, null);
}
```




Chapter

7

Understanding and Using the FAQ Service

This chapter provides a brief description of the Frequently Asked Questions (FAQ) service, the Genesys Knowledge Management features (which create the category structure and standard responses), and the Genesys Content Analyzer (which converts the categories and responses into the FAQ object).

Note: An FAQ sample is provided only for Java in this release.

This chapter contains the following sections:

- [Overview, page 81](#)
- [Genesys Knowledge Management, page 82](#)
- [Genesys Content Analyzer, page 82](#)
- [FAQ Objects, page 83](#)
- [Sample FAQ.jar File, page 83](#)

Overview

The FAQ service allows customers to submit a question, then receive a list of frequently asked questions (FAQs) that relate to their question. Along with the list of FAQs, the response includes a number indicating the confidence with which each FAQ is related to the customer's question. Customers have two search options:

- Search all the categories for FAQs related to their topic, in order to receive a broad range of results.
- Search a specific category, in order to narrow down the results.

However, a customer might not have a specific question. In this case, he or she has two *browse* options:

- Browse through the entire list of FAQs for every category.
- Browse through the list of FAQs for a specified category.

The FAQ list is created by the Genesys Content Analyzer, which is an optional enhancement to Knowledge Management. The remainder of this chapter gives a brief description of Knowledge Management. For further details, see the *eServices 8.0 User's Guide*.

Genesys Knowledge Management

Knowledge Manager functionalities fall into the following four groups:

- **Categories/standard responses/field codes.** A system of categories, organized in a tree structure, provides the means of organizing *standard responses*, which are pre-written responses to interactions. Field codes provide a way to particularize the standard response to individual interactions. Category trees are also integral to the classification functionality of Genesys Content Analyzer (see the third item in this list). You use Knowledge Manager to create category trees, and to create and edit the standard responses and the field codes that they can contain.
- **Screening rules.** Screening rules perform pattern matching on incoming interactions. The results of the pattern matching are then available for use in subsequent steps in routing and in interaction workflows. You use Knowledge Manager to create and edit the screening rules.
- **Genesys Content Analyzer.** This optional functionality uses natural-language processing to analyze incoming interactions and assign them to categories in a category tree. Content analysis uses *models*, which are statistical representations of category trees. Models are produced by *training* on a collection of pre-categorized e-mails. Knowledge Manager controls the training process and displays information about models.
- **FAQ.** With Content Analyzer, you can convert your category structure and standard responses into an FAQ list. You can either post the resulting FAQ list as text on your web site, or use it as the source for an automatic question-answering facility.

Genesys Content Analyzer

Genesys Content Analyzer is an optional enhancement to Genesys E-mail, requiring an additional license. It adds natural-language processing technology to Genesys Knowledge Management.

Models Genesys Content Analyzer applies a classification model—a statistical representation of a category tree—to an incoming e-mail and produces a list of

the categories that the e-mail is most likely to belong to. Each likely category is assigned a percentage rating, indicating the probability that the e-mail belongs to this category.

Training Objects The process of creating a model is called *training*. Training operates on a *training object*, which is a category tree plus a set of text objects. Each text object is assigned to one category in the tree.

Import and Export You can import and export training objects and models.

Components Genesys Content Analyzer does not have components, as such. Rather, it adds functionality to the components of Genesys Knowledge Management:

- It activates Training Server, which has no function in the basic Genesys Knowledge Management, but is required for training models.
- It enables Classification Server to categorize incoming interactions, using models.
- It enables Knowledge Manager to control the creation of training objects, classification models, and FAQ objects.

FAQ Objects

From the FAQ object, you can produce a .jar file, which can in turn be used to:

- Build a Web application that accepts written requests and, using content analysis, returns a set of standard responses.
- Present the contents of the standard response library (or a selection from those contents) as answers to frequently asked questions.

An FAQ object combines a category tree, a training object based on the tree, and optionally, a model built from the training object. The model is required in order to build a Web application.

Sample FAQ.jar File

The supplied, sample FAQ.jar file demonstrates how an FAQ object can present a question/answer list.



Chapter

8

Understanding and Using the Web Callback Service

This chapter describes the eService's Web Callback Service and the Web Callback API in the following sections:

- [Overview, page 85](#)
- [Architecture, page 85](#)
- [Web Callback API, page 86](#)

Overview

This option allows callers to request, cancel, or reschedule a web callback from an agent. Callers will also be able to view a single web callback request or their entire list of request. Contact center managers may choose to send callers an e-mail confirmation after they make the web callback request. Callers can then cancel or reschedule their web callback request using links found in the confirmation e-mail.

The eServices's Web Callback Service allows you to write software that takes advantage of the Web Callback API. Web applications at the contact center will use the Web Callback API to handle communications with Interaction Server.

The Web Callback Sample, see “Web Callback Sample” on [page 144](#) is created using the classes and methods contained within the Web Callback API.

Architecture

Figure 19 on [page 86](#) illustrates the Web Callback components and their relationships. The eServices Web Callback Service relies on Interaction Server to transmit web callback requests to agents. Optional e-mail notifications are

handled by E-mail Server and the optional interaction history is handled by Universal Contact Server (UCS).

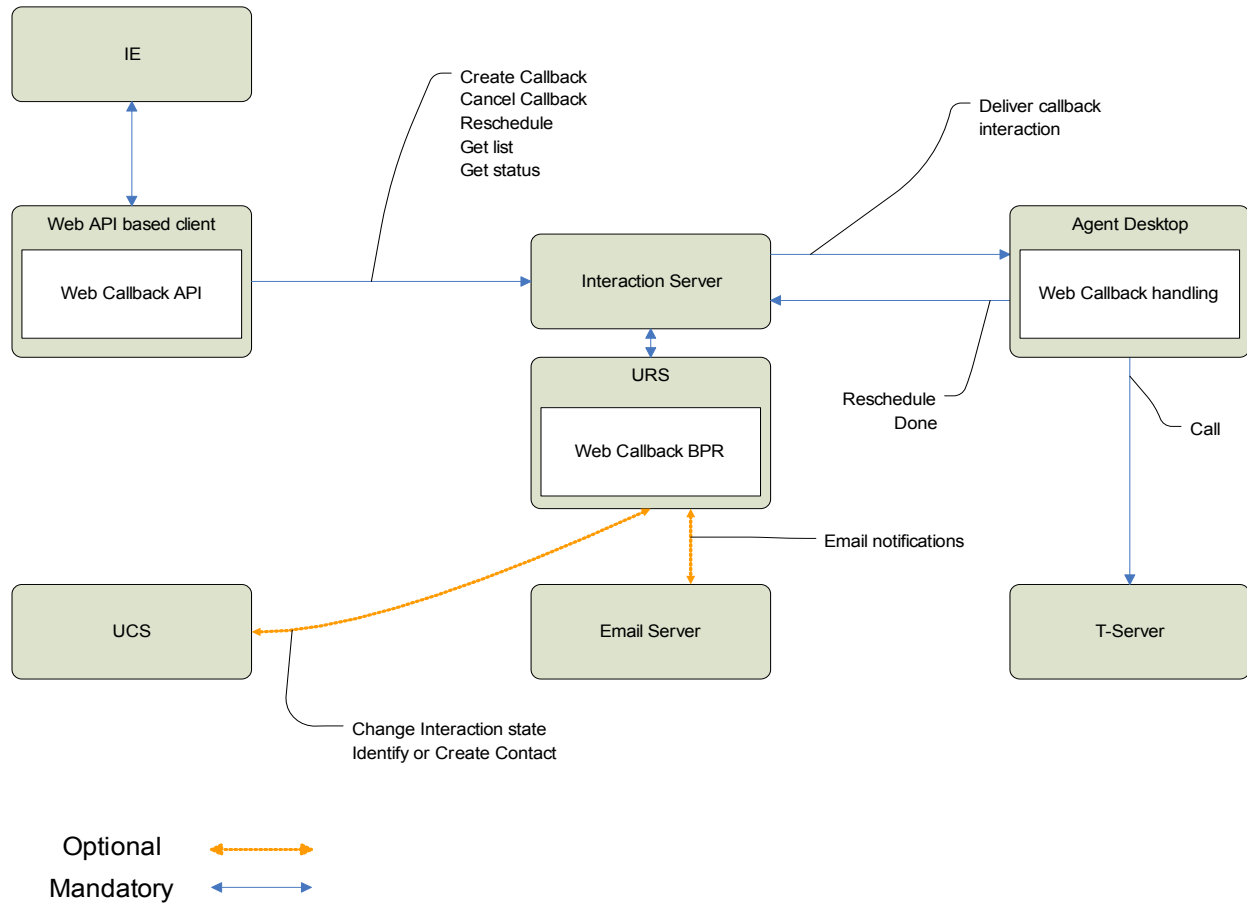


Figure 19: Web Callback Components Architecture

Web Callback API

This section focuses on the five main request types found in the Web Callback API and used by the Web Callback Sample:

- WCBRequestSubmit
- WCBRequestCancel
- WCBRequestReschedule
- WCBRequestGetSingleInteraction
- WCBRequestGetMultipleInteractions

[Figure 20](#) depicts the classes and their associated attributes of the Web Callback API.

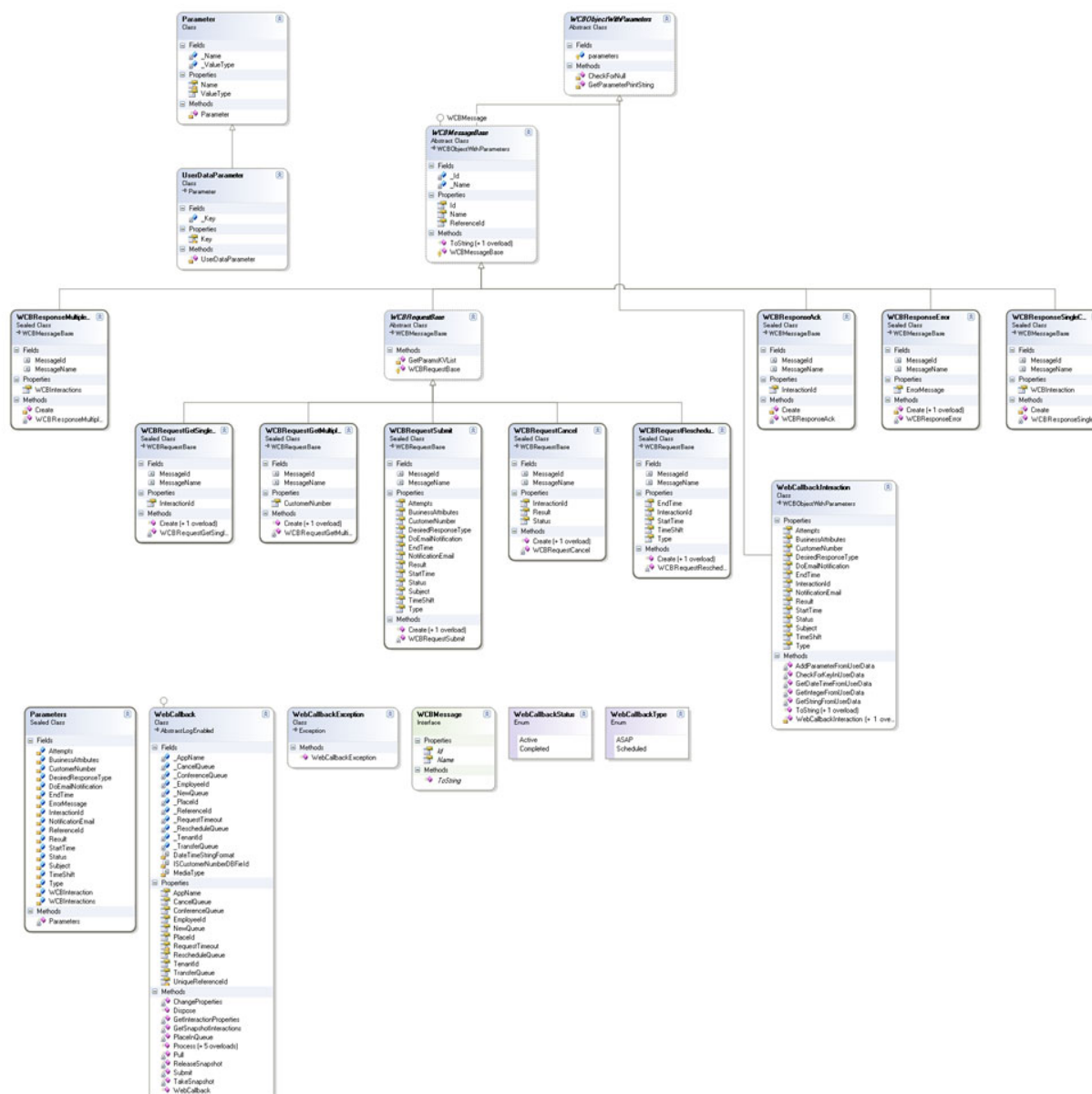


Figure 20: Web Callback API Class Diagram

WCBRequestSubmit

The WCBRequestSubmit class creates a new web callback request. Figure 21 on [page 88](#) shows the WCBRequestSubmit class, its properties, and methods. The properties are explained in detail in Table 7 on [page 88](#).

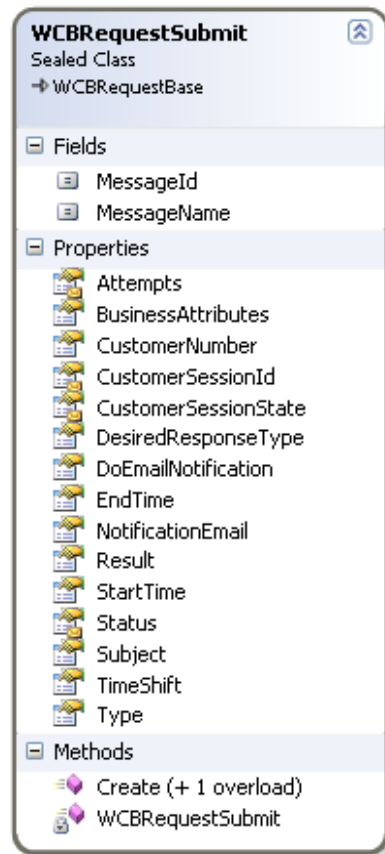


Figure 21: WCBRequestSubmit

Table 7: Properties of the WCBRequestSubmit Class

Name	Type	Details	Default Value
CustomerNumber	String	Mandatory	N/A
BusinessAttributes	Genesyslab.Platform.Commons.Collections.KeyValueCollection	Optional	Empty KVlist
Type	Genesyslab.WebApi.WebCallback.WebCallbackType	Optional	0 – ASAP
Subject	String	Optional	Empty string
StartTime	System.DateTime	Optional	Current time
EndTime	System.DateTime	Optional	Current time + 20 minutes
TimeShift	int	Optional	0

Table 7: Properties of the WCBRequestSubmit Class (Continued)

Name	Type	Details	Default Value
Result	String	Optional	Empty string
DesiredResponseType	String	Optional	Voice
DoEmailNotification	Boolean	Optional	False
NotificationEmail	String	Optional	Empty string

The upper part of [Figure 22](#) shows a successful WCBRequestSubmit sequence flow from the Web Callback Sample to the Interaction Server.

The Web Callback Sample submits a WCBRequestSubmit request to the Web Callback API. The Web Callback API sends both a RequestSubmit and a RequestPlaceInQueue to the Interaction Server. The Interaction Server sends back acknowledgement events to the Web Callback API. A WCBResponseAck is then passed on to the Web Callback Sample.

The lower section of [Figure 22](#) shows the sequence flow when the request is unsuccessful.

Expected responses include:

WCBResponseAck—The Web Callback request was submitted to the Interaction Server successfully.

WCBResponseError—The Web Callback request was unsuccessfully submitted.

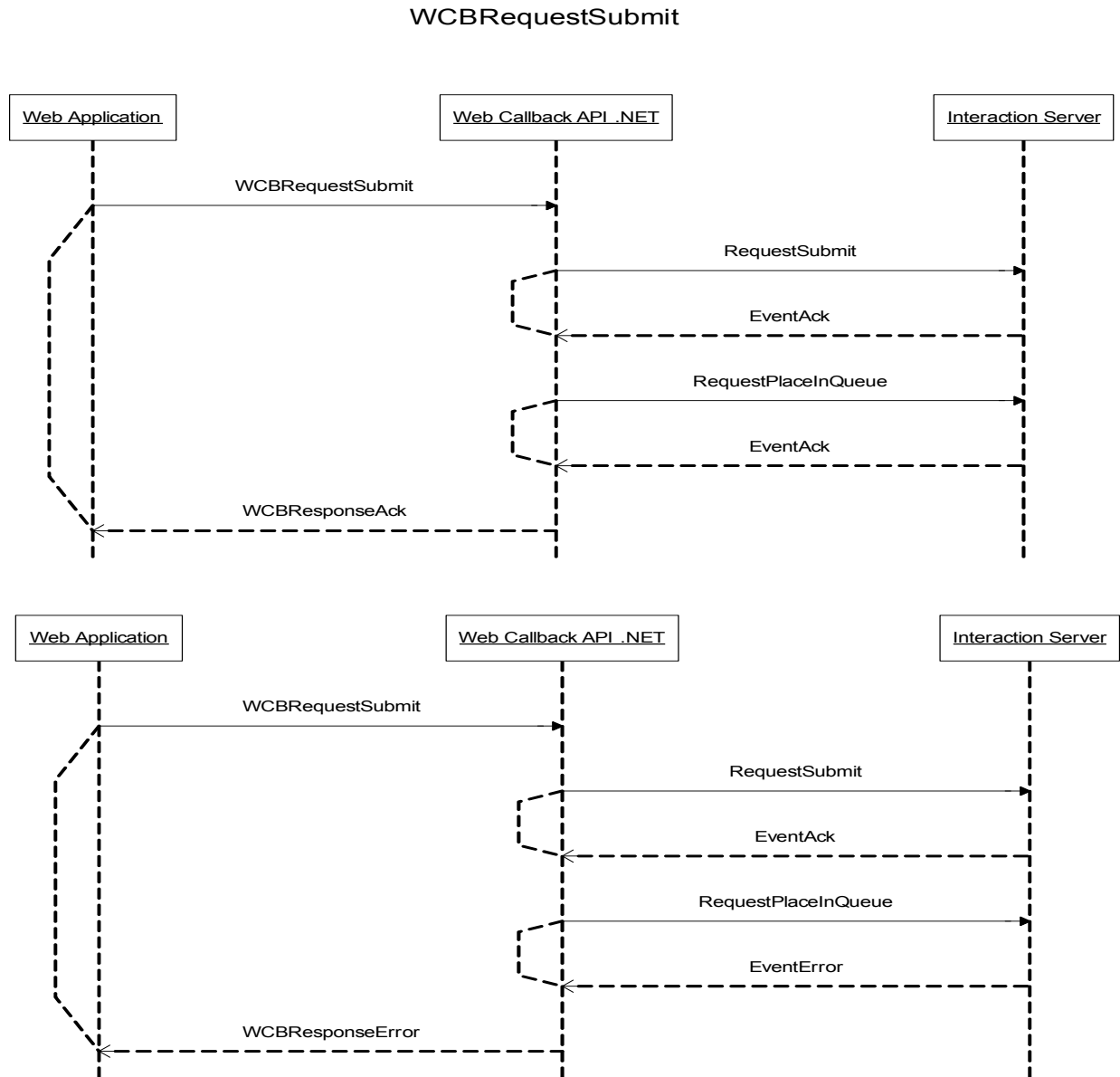
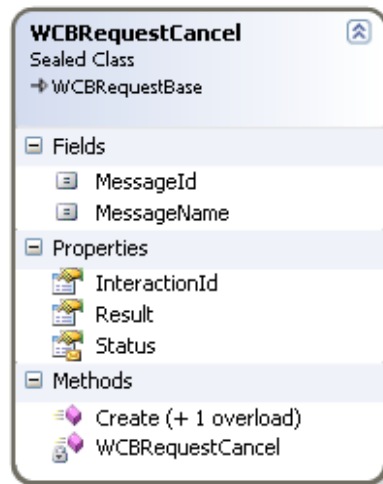


Figure 22: Sequence Flow of a Web Callback Request

WCBRequestCancel

The `WCBRequestCancel` class cancels an existing web callback request. Figure 23 on [page 91](#) shows the `WCBRequestCancel` class, its properties, and methods. The properties are explained in detail in Table 8 on [page 91](#).

**Figure 23: WCBRequestCancel****Table 8: Properties of the WCBRequestCancel**

Name	Type	Details	Default Value
InteractionID	String	Mandatory	N/A
Result	String	Optional	Empty string

The upper part of [Figure 24](#) shows the successful WCBRequestCancel sequence flow from the Web Callback Sample to the Interaction Server.

The Web Callback Sample submits a WCBRequestCancel request to the Web Callback API. The Web Callback API sends RequestGetInteractionProperties, RequestPull, RequestChangeProperties, and RequestPlaceInQueue to the Interaction Server. The Interaction Server sends back EventInteractionProperties, EventPulled, and acknowledgement events to the Web Callback API. A WCBResponseAck is then passed on to the Web Callback Sample.

The lower section of [Figure 24](#) shows the sequence flow when the request is unsuccessful.

Expected responses include:

WCBResponseAck—The Web Callback request was successfully cancelled.

WCBResponseError—The Web Callback request was not cancelled.

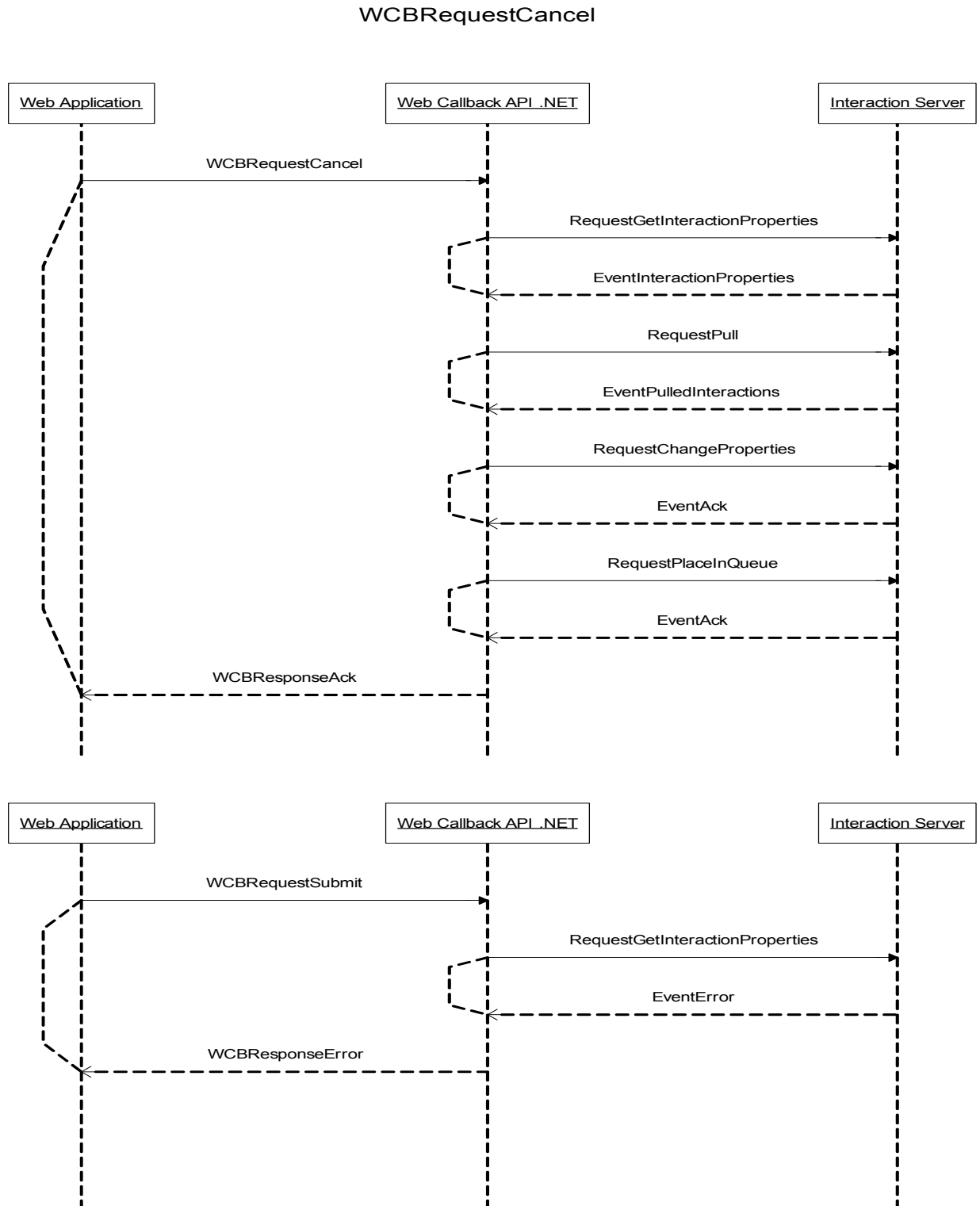


Figure 24: Sequence Flow of a Web Callback Cancellation

WCBRequestReschedule

The WCBRequestReschedule class reschedules an existing web callback request. [Figure 25](#) shows the WCBRequestReschedule class, its properties, and methods. The properties are explained in detail in [Table 9](#).

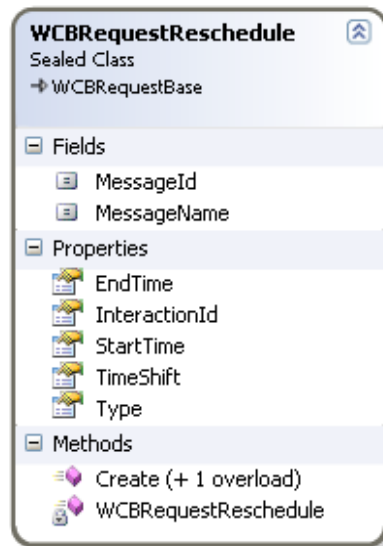


Figure 25: WCBRequestReschedule

Table 9: Properties of the WCBRequestReschedule

Name	Type	Details	Default Value
InteractionID	String	Mandatory	N/A
Type	Genesyslab.WebApi.WebCallback.WebCallbackType	Optional	0 – ASAP
StartTime	System.DateTime	Optional	Current time
EndTime	System.DateTime	Optional	Current time + 20 minutes
TimeShift	int	Optional	0

The upper part of [Figure 26](#) shows the successful WCBRequestReschedule sequence flow from the Web Callback Sample to the Interaction Server.

The Web Callback Sample submits a WCBRequestReschedule request to the Web Callback API. The Web Callback API sends RequestGetInteractionProperties, RequestPull, RequestChangeProperties, and RequestPlaceInQueue to the Interaction Server. The Interaction Server sends back EventInteractionProperties, EventPulled, and acknowledgement events to

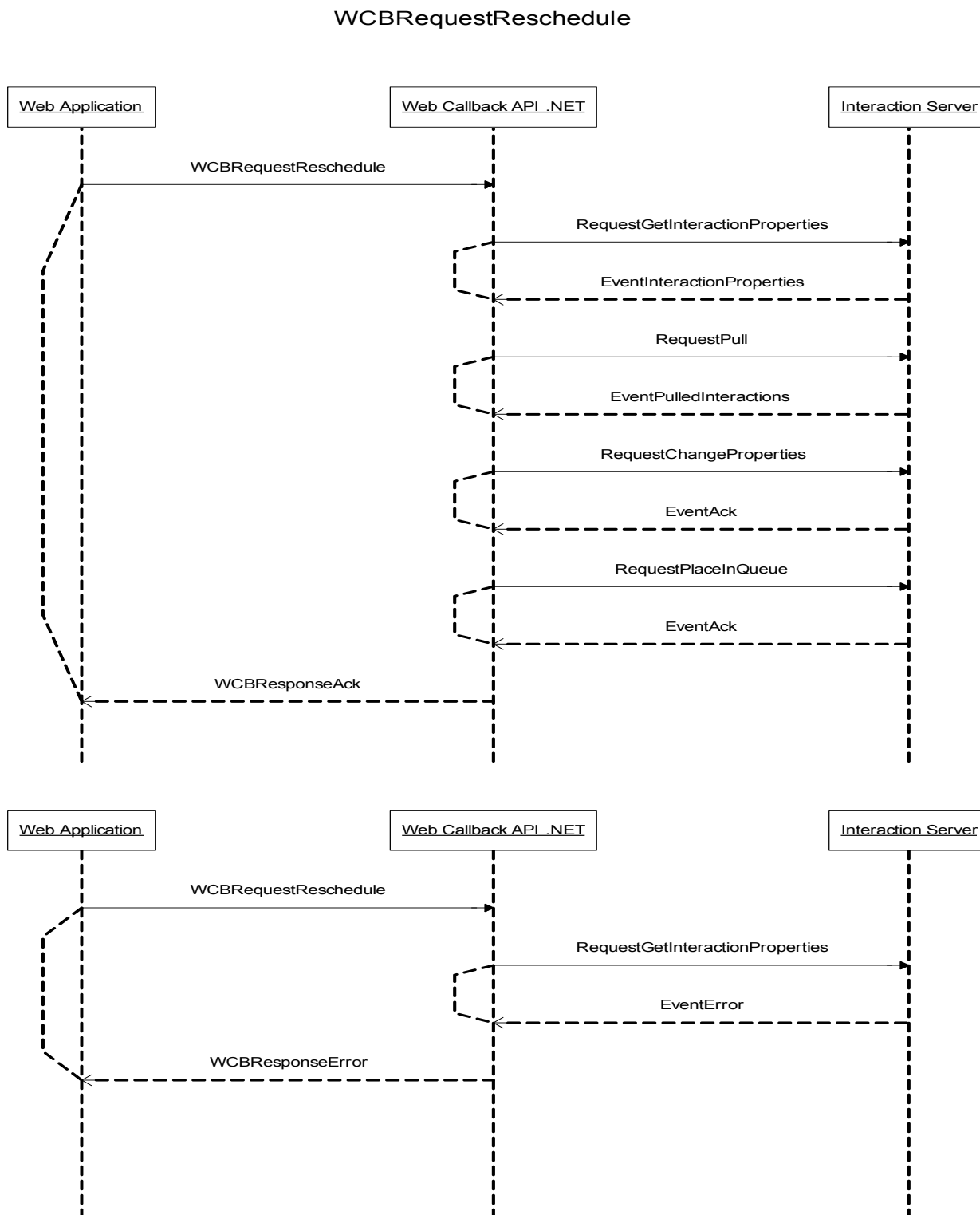
the Web Callback API. A `WCBResponseAck` is then passed on to the Web Callback Sample.

The lower section of [Figure 26](#) shows the sequence flow when the request is unsuccessful.

Expected responses include:

`WCBResponseAck`—The Web Callback request was successfully rescheduled.

`WCBResponseError`—The Web Callback request was not rescheduled.

**Figure 26: Sequence Flow of a Web Callback Reschedule**

WCBRequestGetSingleInteraction

The WCBRequestGetSingleInteraction class retrieves a single web callback request. [Figure 27](#) shows the WCBRequestGetSingleInteraction class, its properties, and methods. The properties are explained in detail in [Table 10](#).

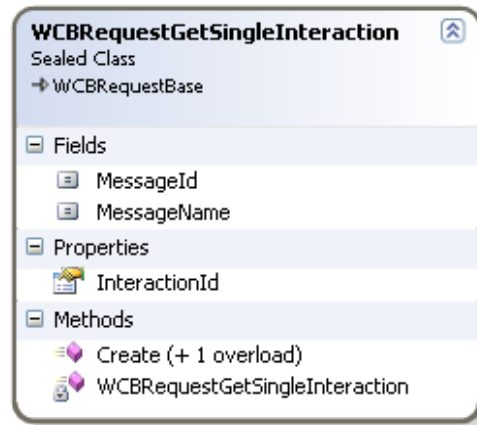


Figure 27: WCBRequestGetSingleInteraction

Table 10: Properties of the WCBRequestGetSingleInteraction

Name	Type	Details	Default Value
InteractionID	String	Mandatory	N/A

The upper part of [Figure 28](#) shows the successful WCBRequestGetSingleInteraction sequence flow from the Web Callback Sample to the Interaction Server.

The Web Callback Sample submits a WCBRequestGetSingleInteraction request to the Web Callback API. The Web Callback API sends RequestGetInteractionProperties to the Interaction Server. The Interaction Server sends back EventInteractionProperties to the Web Callback API. A WCBResponseSingleInteraction is then passed on to the Web Callback Sample.

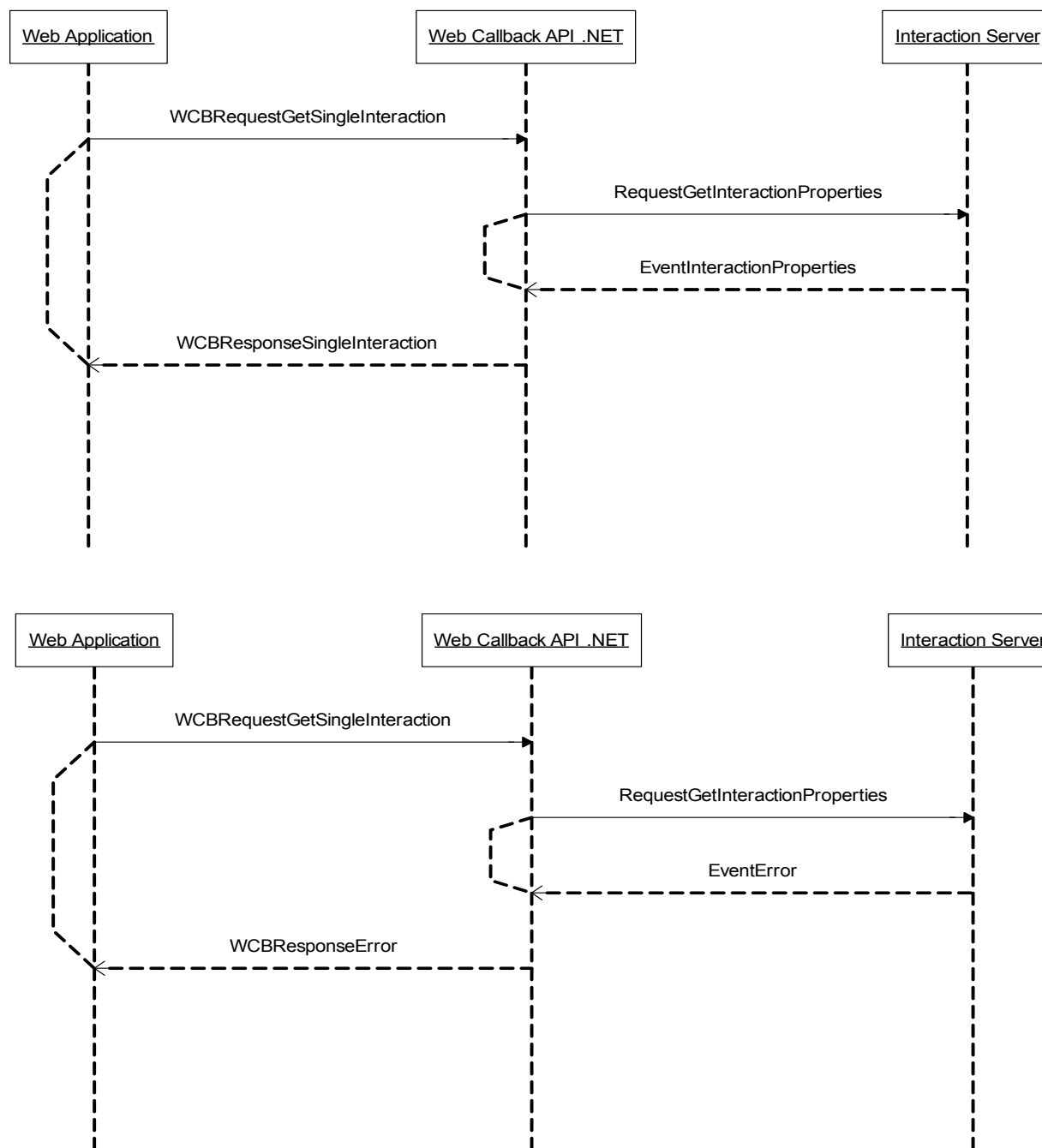
The lower section of [Figure 28](#) shows the sequence flow when the request is unsuccessful.

Expected responses include:

WCBResponseSingleInteraction—The Web Callback request was successfully retrieved.

WCBResponseError—The Web Callback request was not retrieved.

WCBRequestGetSingleInteraction

**Figure 28: Sequence Flow of a Web Callback Retrieve a Single Transaction**

WCBRequestGetMultipleInteractions

The WCBRequestGetMultipleInteractions class retrieves multiple web callback requests. [Figure 29](#) shows the WCBRequestGetMultipleInteractions class, its properties, and methods. The properties are explained in detail in [Table 11](#).

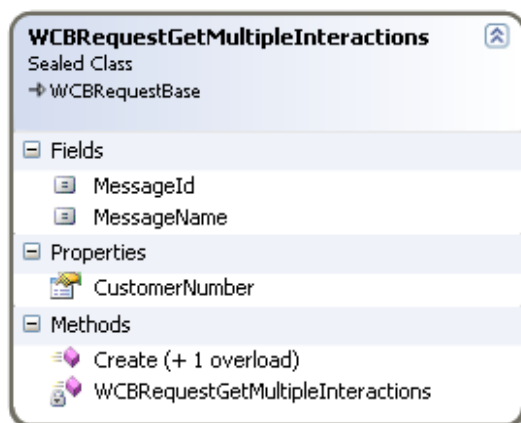


Figure 29: WCBRequestGetMultipleInteractions

Table 11: Properties of the WCBRequestGetMultipleInteractions

Name	Type	Details	Default Value
CustomerNumber	String	Mandatory	N/A

The upper part of [Figure 30](#) shows the successful WCBRequestGetMultipleInteractions sequence flow from the Web Callback Sample to the Interaction Server.

The Web Callback Sample submits a WCBRequestGetMultipleInteractions request to the Web Callback API. The Web Callback API sends the request onto the Interaction Server where it is placed in the queue. The Interaction Server sends back acknowledgements to the Web Callback API where they are passed onto the Web Callback Sample.

The Web Callback API sends RequestTakeSnapshot, RequestGetSnapshotInteractions, and RequestReleaseSnapshot to the Interaction Server. The Interaction Server sends back EventSnapshotTaken, EventSnapshotInteractions, and EventAck to the Web Callback API. A WCBResponseMultipleInteractions is then passed on to the Web Callback Sample.

The lower section of [Figure 30](#) shows the sequence flow when the request is unsuccessful.

Expected responses include:

WCBResponseMultipleInteractions—The Web Callback requests were successfully retrieved.

WCBResponseError—The Web Callback requests were not retrieved.

WCBRequestGetMultipleInteractions

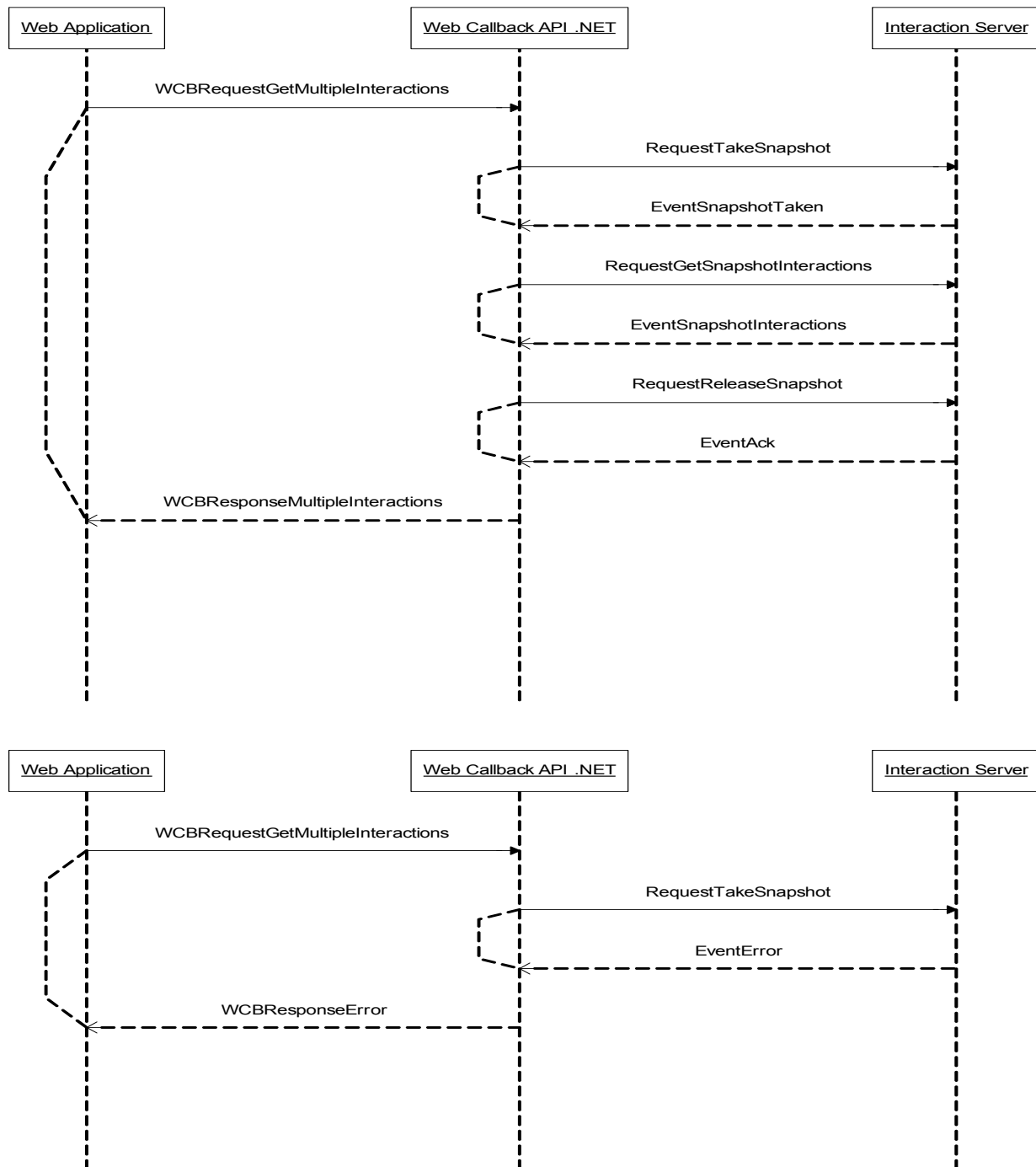


Figure 30: Sequence Flow of a Web Callback Retrieve Multiple Transactions



Chapter

9

eServices Samples for Java

This chapter examines eServices's web-based samples for Java, as well as their code. (.NET developers should instead see Chapter 10, "eServices Samples for .NET," on [page 225](#).)

This chapter covers the following topics:

- [Overview, page 102](#)
- [Shared Files, page 104](#)
- [E-Mail Sample, page 108](#)
- [E-Mail with Attachments Sample, page 113](#)
- [Chat Sample, page 121](#)
- [Advanced Chat, page 132](#)
- [Chat Widget Sample, page 134](#)
- [Survey Sample, page 138](#)
- [Web Callback Sample, page 144](#)
- [Interaction Submit \(Genesys 3rd Party Media\) Sample, page 151](#)
- [Statistical Information Sample, page 160](#)
- [Universal Contact Server Sample, page 166](#)
- [Dynamic Invitation Sample, page 181](#)
- [FAQ \(Web Self-Serve\) Sample, page 187](#)
- [Media Availability Sample, page 193](#)
- [Cobrowse Samples Overview, page 197](#)
- [Basic Cobrowse Sample, page 197](#)
- [Chat and Cobrowse Sample, page 203](#)
- [Cobrowse with Meet Me, page 211](#)
- [Cobrowse with Initial Startup Page, page 214](#)
- [Cobrowse with Dynamic Startup Page Sample, page 218](#)

Overview

This chapter explains—through a review of the key functions in the respective samples—how to implement chat, e-mail, web-collaboration, web-callback, Genesys 3rd Party Media, FAQ, and proactive web-engagement functions in your web application. The samples come with the eServices Interactive Management CD. See Chapter 2, “About the Samples,” on [page 45](#) for information on installation of the samples.

Please note the following about the sample code that is presented and organized in this chapter:

- The code is excerpted from the actual sample code, because the actual code is too long to be displayed here.
- The code that is excerpted illustrates a point or calls attention to a particular feature. You should refer to the actual code and to the *Platform SDK 8.0 API Reference* for the particular SDK that you are using, which provides the authoritative information on methods, functions, and events for your SDK.
- Although this chapter reviews the different functions that the sample performs, some of these functions and the code that is excerpted might not be presented in the same order or layout as in the sample.

Sample Overview

The following web-based samples are discussed in this chapter:

- E-mail—This sample demonstrates how to use the Platform SDK Java API and Java Server Pages (JSPs) to send e-mail by using a web form.
- E-mail with Attachments—This sample demonstrates how to use the Platform SDK Java API and JSPs to send an e-mail that contains an attachment using a web form.
- Chat—This sample demonstrates how to use the Platform SDK Java API for the Flex Chat protocol and JSPs to implement a chat feature on a web form. It includes “user-typing” notification functionality.
- Advanced Chat—This sample demonstrates how to use the Platform SDK Java API and JSPs to implement a chat feature on a web form. It includes smiles and hyperlink support. This sample also demonstrates how to include a survey in your chat window.
- Chat Widget—This sample demonstrates how to use the Platform SDK Java API and JSPs to customize a chat window.
- Web Callback—This sample demonstrates how to use the Web Callback API to implement web callback.
- Interaction Submit (Genesys 3rd Party Media)—This sample demonstrates how to use the Platform SDK Java API and JSPs to submit interactions to—and update interactions on—Interaction Server by using a web form.

- **Statistical Information**—This sample demonstrates how to use the Platform SDK Java API and JSPs to select predefined statistics and retrieve their current value by using a web form.
- **Universal Contact Server**—This sample demonstrates how to use the Platform SDK Java API and JSPs to communicate with UCS to access the interaction history of a contact by using a web form.
- **Dynamic Invitation**—This sample demonstrates how to use the Platform SDK Java API and JSPs to analyze the activities of a client on a web page and enables you to offer to chat with the client.
- **FAQ (Web Self-Serve)**—This sample demonstrates how to use the Knowledge Server FAQ functionality on a web form.
- **Media Availability**—This sample demonstrates how to check which media are currently available.
- **Basic Cobrowse**—This sample demonstrates how to use the Cobrowsing Server API to implement cobrowse on a web form.
- **Chat and Cobrowse**—This sample demonstrates how to use the Cobrowsing Server API to implement a chat feature with cobrowse on a web form.
- **Cobrowse with Meet Me**—This sample demonstrates how to use the Cobrowsing Server API to implement the “meet me” feature for cobrowse on a web form.
- **Cobrowse with Initial Startup Page**—This sample demonstrates how to use the Cobrowsing Server API to implement an initial startup page for cobrowse on a web form.
- **Cobrowse with Dynamic Startup Page**—This sample demonstrates how to use the Cobrowsing Server API to implement a dynamic startup page for cobrowse on a web form.

Figure 31 on [page 104](#) shows the SimpleSamples802 directory structure of the Java version of the samples.

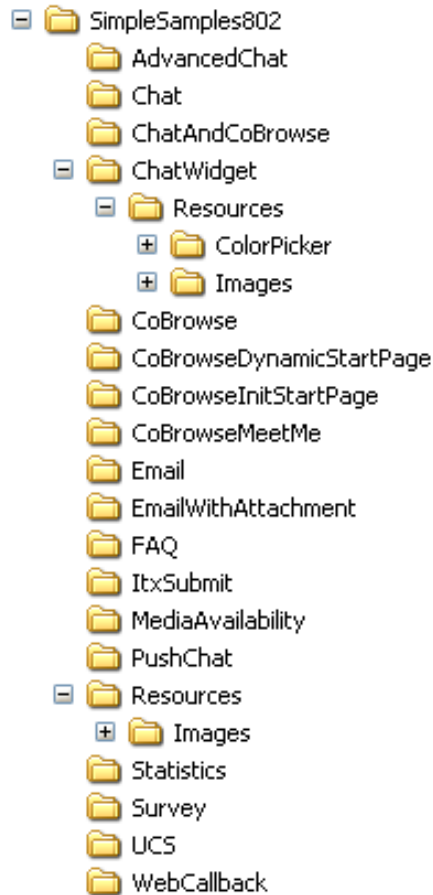


Figure 31: Directory Structure

Shared Files

The following files are used in the Java web samples. The following subsections group these files into categories and explain them in detail.

- `CommLib.js`—See “Common JavaScript Functions” on [page 105](#).
- `constants.jsp`—See “Constants” on [page 105](#).
- `Security.jsp`—See “Masking Text” on [page 107](#).
- `index.htm`—See “Greetings Page” on [page 107](#).
- `Left.htm`—See “Greetings Page” on [page 107](#).
- `Main.htm`—See “Greetings Page” on [page 107](#).
- `Top.htm`—See “Greetings Page” on [page 107](#).
- `fon.gif`—See “Graphics” on [page 107](#).
- `genesyslogo-trans.gif`—See “Graphics” on [page 107](#).

File Descriptions

Common JavaScript Functions

The `CommLib.js` file stores common functions that all the samples use. It contains functions that:

- Identify a user's web browser.
- Retrieve the handle to HTML frames or control objects.
- Perform basic string manipulation and encoding.
- Retrieve submitted form parameters and return the current time.

Constants

The `constants.jsp` file (see [Table 12](#)) contains values such as:

- An e-mail user's first name, last name, phone number, and e-mail address.
- The timeframe window in which to call a user back.
- The media and routing information to use for a request.
- Load balancing information.
- Chat-refresh timeout.

Table 12: Constant Values in the `constants.jsp` File

Variable Name	Default Value	Description
<code>fldnPhoneNumber</code>	<code>PhoneNumber</code>	User's phone number.
<code>fldnFirstName</code>	<code>FirstName</code>	User's first name.
<code>fldnLastName</code>	<code>LastName</code>	User's last name.
<code>fldnEmailAddress</code>	<code>EmailAddress</code>	User's e-mail address.
<code>fldnFromAddress</code>	<code>FromAddress</code>	E-mail's "from" address.
<code>fldnStartTime</code>	<code>StartTime</code>	Start time.
<code>fldnEndTime</code>	<code>EndTime</code>	End time.
<code>fldnMedia</code>	<code>Media</code>	Communication medium such as chat or e-mail.
<code>fldnNow</code>	<code>Now</code>	Current time.
<code>fldnRouteInfo</code>	<code>RouteInfo</code>	Routing point information.
<code>fldnSubject</code>	<code>Subject</code>	Subject of an e-mail message.
<code>fldnEmailBody</code>	<code>EmailBody</code>	Body of an e-mail message.

Table 12: Constant Values in the constants.jsp File (Continued)

Variable Name	Default Value	Description
fldnReplyFrom	Mailbox	Sender of an e-mail message.
strTenant	TENANT_NAME value	Name of the tenant.
strEmailQueue	Inbound queue	E-mail queue.
strChatQueue	Chat inbound queue	Chat queue.
strQueueKey	“default”	Queue key.
strChatStatInterval	“”	Chat statistics interval.
strEmailStatInterval	“”	E-mail statistics interval.
strConavServerUrl	CONAV_URL	Host name of the default cobrowse server—for example, cobrowse.yourdomain.com.
strConavChannelID	“Default”	Cobrowse server might have different channels processing requests with different settings. This is the name of the channel that is used for communication with cobrowse server.
chatRefreshTimeout	5000	Frequency at which a chat session is refreshed.
strStatTimeProfile	“Default”	Stat sample constant.
strStatTimeInterval	Range 0-120	Stat sample constant.
Web Callback Constants		
wcbNewQueue	“New”	Queue where all the newly created interactions are placed.
wcbCancelQueue	“Stop by Customer”	Queue where all the cancelled interactions are placed.
wcbConferenceQueue	“Queue for Conference”	Queue where Agent places web callback interactions during a conference.
wcbTransferQueue	“Queue for Transfer”	Queue where an Agent places web callback interactions during a transfer.
wcbRescheduleQueue	“Reschedule by Customer”	Queue where all the rescheduled interactions are placed.
wcbPlaceId	“pWebCallback”	Name of the Place Id where an Agent used by WebCallback API will login.

Masking Text

The `Security.jsp` file contains a `mask_html()` method (with different signatures) that translates special characters such as `"`, `'`, `<`, `>`, and `&` into HTML abbreviations such as `<`, `>`, and `"`.

Presentation

Greetings Page

The `index.htm`, `Top.htm`, `Left.htm`, and `Main.htm` files are used to create the main or greeting page for access to the web samples. These files present an HTML table menu that contains links to each of the samples.

Graphics

The `fon.gif` and `genesyslogo-trans.gif` files are graphics files that are used in `index.htm` and various samples. The `fon.gif` file is a graphics file for background-wallpaper design. The `genesyslogo-trans.gif` file contains the Genesys company logo. The Cascading Style Sheet (.css) file and additional graphics that are used in these samples are found in the `.../resource` directory. See Figure 31, “Directory Structure,” on [page 104](#) for details.

Common Code

Many of the samples contain some common code. The following subsections explain some of the common code that is found in each of the samples.

Setting the Content Type and Character Encoding

The first line in most samples is a call to the `response.setContentType()` method. This sets the content type or character encoding to use in the response to the client. The code retrieves this value under the `Options` tab for the required server `Application` object in `Configuration Server`:

```
<%response.setContentType("text/html; charset=" + i18nsupport.GetCharSet());%>
```

Loading Libraries and Importing Files

The sample code then loads the required libraries into memory. Check each sample to determine which libraries are required:

```
<%@ page import="com.genesyslab.platform.commons.protocol.Endpoint" %>
<%@ page import="com.genesyslab.platform.commons.protocol.Message" %>
<%@ page import="com.genesyslab.platform.webmedia.protocol.EmailProtocol" %>
<%@ page import="com.genesyslab.platform.webmedia.protocol.email.EmailProperties" %>
<%@ page import="com.genesyslab.platform.webmedia.protocol.email.events.EventAck" %>
<%@ page import="com.genesyslab.platform.webmedia.protocol.email.events.EventError" %>
```

```

<%@ page
    import="com.genesyslab.platform.webmedia.protocol.email.requests.RequestSubmit" %>
<%@ page import="com.genesyslab.webapi.core.LoadBalancer" %>
<%@ page import="com.genesyslab.webapi.core.LoadBalancerException" %>
<%@ page import="com.genesyslab.webapi.core.ServiceInfo" %>...

```

The sample must access other JSP files. It includes or imports the `constants.jsp` and `Security.jsp` files:

```

<%@ include file="../constants.jsp" %>
<%@ include file="../Security.jsp" %>Handling Content

```

Creating the HTML Header

All of the samples contain an HTML header tag. They are all very similar except for the `<title>` content:

```

<html>
<head>
    <title>MCR 8.0 samples. E-mail over the Web sample</title>
    <link rel="stylesheet" type="text/css" href="../resource/style/StyleSheet.css"/>
</head>
<body LANGUAGE="javascript" onload="window_onload();" onunload="window_onunload();"
    class="samplesBody" >
<SCRIPT LANGUAGE=javascript SRC="../CommLib.js"></SCRIPT>...

```

Retrieving Parameters

This section of the Java code retrieves the request parameters, such as the first name and last name of a customer, from the `i18nsupport.GetSubmitParametr()` method instead of from the `HTTPServletResponse.getParameter()` method. This step is necessary to support foreign languages. For more information on language support, see “International Language Support” on [page 33](#). Any parameter that returns a null value is set to "":

```

String action = i18nsupport.GetSubmitParametr(request, "action");
String first_name = i18nsupport.GetSubmitParametr(request, fldnFirstName);
if (first_name == null) first_name = "";
String last_name = i18nsupport.GetSubmitParametr(request, fldnLastName);
if (last_name == null) last_name = "";...

```

E-Mail Sample

This section presents the purpose, functionality overview, and code implementation of the E-mail Sample.

Purpose

The E-mail Sample code demonstrates how a user can send an e-mail request via a web form. What sets this sample apart from an ordinary or external e-mail is that the web-form e-mail message goes through E-mail Server, which routes the e-mail message to the appropriate agent by using Genesys Universal Routing Server.

The sample is a JSP file that shows how to:

- Connect to E-mail Server by using the Platform SDK E-mail Protocol.
- Submit an e-mail request to E-mail Server.
- Report any possible errors in the sample.
- Disconnect from E-mail Server.
- Gather submitted information.
- Communicate with the load balancing service.
- Communicate with E-mail Server.
- Generate HTML responses and JavaScript code, based on the responses from E-mail Server.
- Mask all potentially dangerous server-side data or symbols such as <, >, ", and '.

Functionality Overview

The following sections review the code that is used in the implementation of the different e-mail functions:

- “Getting Instances of Load Balancer and E-Mail Server” on [page 110](#)
- “Constructing the HTML Body” on [page 110](#)
- “Connecting to E-mail Server” on [page 111](#)
- “Tracking States and Submission” on [page 112](#)
- “Closing the Connection” on [page 113](#)
- “Handling User Events” on [page 113](#)

For information about code that is common to all of the samples—such as retrieving parameters, setting the content type, character encoding, loading libraries, and importing files—see “Common Code” on [page 107](#).

Files

The `.../Email` directory contains the E-mail Sample. The sample consists of a single file, `Email.jsp`.

Code Explanation

The following subsections explain the code in the `Email.jsp` file.

Getting Instances of Load Balancer and E-Mail Server

This section of code creates an instance of Load Balancer and returns an available instance of E-mail Server for each request. It might seem a bit odd to have this step occur after the parameter- retrieval step. This order occurs because the code actually gets a new instance of Load Balancer and E-mail Server instances for each request. Hence, for every submission, the JSP gets the new objects:

```
String svcHost = "";
int svcPort = -1;
ServiceInfo si = null;
try
{
    si = LoadBalancer.GetServiceInfo(CfgAppType.CFGEmailServer, strTenant);
    svcHost = si.getHost();
    svcPort = si.getPort();}...
```

Constructing the HTML Body

Next, the code includes more HTML code for the creation of input boxes and buttons:

```
<form id=frm_irs name="frm_irs" method="post" action="Email.jsp"
onSubmit="JavaScript:return Submit_onClick (1);">
    <table style="width:100%" cellpadding="2">
        <tr>
            <td class="samplesSampleTopToolbar">
                <table >
                    <tr >
                        <td style="width:617px"
class="samplesSampleNameTitleWhite">Email over the Web sample</td>
                        <td style="width:auto" ></td>
                    </tr>
                </table>
            </td>
        </tr>

        <tr>
            <td class="samplesSampleParagraph">
                Sample of Email submission over the web based on the PSDK Java API
                classes and JSP pages.
            </td>
        </tr>
        <tr>
            <td>
```

```

        <table style="width:617px" border="0" >
        <tr>
            <td class="samplesSampleFieldTitle" ></td>
            <td></td>
            <td style="width:6px; vertical-align: middle;"></td>
        </tr>

        <tr>
            <td class="samplesSampleFieldTitle" >First name:</td>
            <td><input class="samplesFieldStyle" type="text"
name="<%=mask_html(fldnFirstName)%>" value="<%=mask_html(first_name)%>"></td>
            <td style="width:6px; vertical-align: middle;"><font
class="samplesMandatoryMark">*</font></td>
        </tr>

        <tr>
            <td class="samplesSampleFieldTitle" >Last name:</td>
            <td><input class="samplesFieldStyle" type="text"
name="<%=mask_html(fldnLastName)%>" value="<%=mask_html(last_name)%>"></td>
            <td style="width:6px; vertical-align: middle;"><font
class="samplesMandatoryMark">*</font></td>
        </tr>
        <tr>...

```

Connecting to E-mail Server

Next, the Java code attempts to get a connection to E-mail Server. If it does not succeed, an error message is displayed:

```

if (svcHost != null && !svcHost.equalsIgnoreCase("") && svcPort != -1 && action != null
&& !action.equals(""))
{
    EspEmailProtocol email = new EspEmailProtocol(new Endpoint(new URI("tcp://" +
svcHost + ":" + String.valueOf(svcPort))));
    boolean bConnected = false;
    try
    {
        email.open();
        bConnected = true;
    }
    catch (Exception e)
    {
        %>
        <script language="javascript">
ServerMessage.innerHTML = "We are sorry, Email Server is unavailable at this time.
Please try again later.";
        </script>
        %>
    }
}

```

Tracking States and Submission

If the connection is successful and the Submit button was clicked, the properties and contents of the e-mail message are submitted in a request to E-mail Server:

Note: The sample JSP monitors the *state* of the file, based on which button the user selects.

```
if (bConnected && action.equals("Submit"))
{
    RequestCreateWebEmailIn espRequest = RequestCreateWebEmailIn.create();
    espRequest.setFirstName(first_name);
    espRequest.setLastName(last_name);
    espRequest.setFromAddress(email_address);
    espRequest.setText(body);
    espRequest.setSubject(subject);
    if (reply_from != null && !reply_from.equalsIgnoreCase(""))
        espRequest.setMailbox(reply_from);

    Message msg = email.request(espRequest, 30000);
```

An appropriate message will be displayed, based on whether the submission was successful:

```
else if (msg instanceof EventCreateWebEmailIn)
{
    EventCreateWebEmailIn eventAck = (EventCreateWebEmailIn) msg;
    id = eventAck.getNewInteractionId();

%>
<script language="javascript">
ServerMessage.innerHTML = "Request (<%=mask_html(id)%>) has been successfully submitted
    to Email Server.";
</script>
<%
    }
    else if (msg instanceof EventError)
    {
        EventError eventError = (EventError) msg;

%>
<script language="javascript">
ServerMessage.innerHTML = "Could not submit the email to Email Server. Reason:
    <%=mask_html(eventError.getFaultString())%>";
</script>
<%
```

Closing the Connection

The connection to E-mail Server must be closed. Otherwise, you will have orphan connections to the server that do not relinquish resources— principally, network resources (such as sockets) and memory.

```
if (email != null && bConnected)
    email.close();
}
```

Handling User Events

Four JavaScript functions handle user events:

- `window_onload()`—Sets the request ID for this form. If the document is not undefined, it sets the value of the form ID to the value of the request ID.
- `window_onunload()`—Has no code. It is an empty function.
- `Submit_onClick()`—Sets the action value to `Submit`. Because this is a real submission from the user and not a JSP workaround, the function must verify that the e-mail address and sender data are in an acceptable format. If the function finds invalid characters, it presents a dialog box and requests that the user correct the problem. When the data is acceptable, the function calls the HTML `submit()` function.
- `Reset_onClick()`—Calls the `reset()` method of the form to clear out all of the data that is entered.

E-Mail with Attachments Sample

This section presents the purpose, functionality overview, and code implementation of the E-mail with Attachments Sample.

Purpose

The E-mail with Attachments Sample code demonstrates how a user can send an e-mail message that contains an attachment by using a web form.

Functionality Overview

The following sections review the code that is used in the implementation of the different E-mail with Attachments functions:

- “Declaring Variables” on [page 114](#)
- “Uploading the Multipart File” on [page 115](#)
- “Getting Load Balancer and E-Mail Server Instance” on [page 116](#)
- “Constructing the HTML Body” on [page 117](#)
- “Connecting to E-mail Server” on [page 118](#)

- “Tracking States and Submission” on [page 119](#)
- “Closing the Connection” on [page 120](#)
- “Handling User Events” on [page 120](#)

For information about code that is common to all of the samples— such as retrieving parameters, setting the content type, character encoding, loading libraries, and importing files—see “Common Code” on [page 107](#).

Files

The `.../EmailWithAttachment` directory contains the E-mail with Attachments Sample. The sample consists of a single file; `Email.jsp`.

Code Explanation

The following subsections explain the code in `Email.jsp`.

Note: The `commons-fileupload-1.1.jar` and `commons-io-1.2.jar` Java libraries are used to help handle attachments. You can download them from the Apache website.

Declaring Variables

The following variables are declared:

```
String action = "";
String first_name = "";
String last_name = "";
String email_address = "";
String subject = "";
String body = "";
String reply_from = "";
String id = "";
byte[] attachment1 = null;
byte[] attachment2 = null;
String filename1 = "";
String filename2 = "";
String contenttype1 = "";
String contenttype2 = "";
String strFileUploaderError = "";
boolean isMultipart = FileUpload.isMultipartContent(request);
```

Instances of `FileItemFactory` and `ServletFileUpload` are created. The upload maximum size is also set:

```
FileItemFactory factory = new DiskFileItemFactory();
ServletFileUpload upload = new ServletFileUpload(factory);
upload.setSizeMax(1024 * 1024);
```

Uploading the Multipart File

The following code looks for an attached file to upload:

```
if (isMultipart)
{
    List items = null;
    try
    {
        items = upload.parseRequest(request);
        Iterator itr = items.iterator();

        while (itr.hasNext())
        {
            FileItem item = (FileItem) itr.next();

            // check if the current item is a form field or an uploaded file
            if (item.isFormField())
            {
                // get the name of the field
                String fieldName = item.getFieldName();
                String fieldValue = item.getString(i18nsupport.GetCodePage());

                if (fieldName.equals("action"))
                {
                    action = fieldValue;
                }
                else if (fieldName.equals(fldnFirstName))
                {
                    first_name = fieldValue;
                }
                else if (fieldName.equals(fldnLastName))
                {
                    last_name = fieldValue;
                }
                else if (fieldName.equals(fldnFromAddress))
                {
                    email_address = fieldValue;
                }
                else if (fieldName.equals(fldnSubject))
                {
                    subject = fieldValue;
                }
                else if (fieldName.equals(fldnEmailBody))
                {
                    body = fieldValue;
                }
                else if (fieldName.equals(fldnReplyFrom))
                {
                    reply_from = fieldValue;
                }
            }
        }
    }
}
```

```

        else if (fieldName.equals("id"))
        {
            id = fieldValue;
        }
    }
    else
    {
        String fieldName = item.getFieldName();
        String fileName = item.getName();
        String contentType = item.getContentType();
        boolean isInMemory = item.isInMemory();
        long sizeInBytes = item.getSize();
    }

```

In the following code, the `FilenameUtils.getName()` function is used to return the full filename. Only the text that comes after the last forward slash or backslash will be returned. This function handles both UNIX and Windows file-name formats:

```

if (fileName != null)
{
    fileName = FilenameUtils.getName(fileName);
}

if (fieldName.equals("attachment1"))
{
    filename1 = fileName;
    contenttype1 = contentType;
    attachment1 = item.get();
}
if (fieldName.equals("attachment2"))
{
    filename2 = fileName;
    contenttype2 = contentType;
    attachment2 = item.get();
}...

```

Getting Load Balancer and E-Mail Server Instance

This section of code creates an instance of Load Balancer and returns an available instance of E-mail Server for each request. It might seem a bit odd to have this step occur after the parameter-retrieval step. This order occurs because the code actually gets new instance of Load Balancer and E-mail Server for each request. Hence, for every submission, the JSP gets the new objects:

```

String svcHost = "";
int svcPort = -1;
ServiceInfo si = null;
try
{

```

```

    si = LoadBalancer.GetServiceInfo(CfgAppType.CFGEmailServer, strTenant);
    svcHost = si.getHost();
    svcPort = si.getPort();
}...

```

Constructing the HTML Body

Next, the code includes more HTML code for the creation of input boxes and buttons. The first line of code sets the `enctype` attribute of the form to `multipart/form-data`. The `enctype` attribute determines how the form will be encoded, and setting it to `multipart/form-data` will enable file upload. At the bottom of the form are two fields where file names can be entered and attached to the e-mail message:

```

<form enctype="multipart/form-data" id=frm_irs name="frm_irs" method="post"
  action="Email.jsp" onSubmit="JavaScript:return Submit_onClick (1);">
  <table style="width:100%" cellpadding="0" cellspacing="2">
    <tr>
      <td class="samplesSampleTopToolbar">
        <table >
          <tr >
            <td style="width:617px" class="samplesSampleNameTitleWhite">Email
over the Web sample</td>
            <td style="width:auto" ></td>
          </tr>
        </table>
      </td>
    </tr>

    <tr>
      <td class="samplesSampleParagraph">
        Sample of Email submission over the web based on the PSDK Java API
classes and JSP pages. Allows to attach files with form.
      </td>
    </tr>
    <tr>
      <td>
        <table style="width:617px" border="0" >

          <tr>
            <td class="samplesSampleFieldTitle" ></td>
            <td></td>
            <td style="width:6px; vertical-align: middle;"></td>
          </tr>

          <tr>
            <td class="samplesSampleFieldTitle" >First name:</td>

```

```

        <td><input class="samplesFieldStyle" type="text"
name="<%=mask_html(fldnFirstName)%>" value="<%=mask_html(first_name)%>"></td>
        <td style="width:6px; vertical-align: middle;"><font
class="samplesMandatoryMark">*</font></td>
        </tr>...

...<tr>

        <td class="samplesSampleFieldTitle" >File to upload #1:</td>
        <td>
            <div class="samplesCoolFileInputs">
                <input class="examplesCoolFile" type="file"
name="attachment1">
            </div>
        </td>
        <td></td>
    </tr>

    <tr>
        <td class="samplesSampleFieldTitle" >File to upload #2:</td>
        <td>
            <div class="samplesCoolFileInputs">
                <input class="examplesCoolFile" type="file"
name="attachment2">
            </div>
        </td>
        <td></td>
    </tr>...

```

Connecting to E-mail Server

Before you attempt to connect to E-mail Server, you must check for and display any upload errors:

```

    if (strFileUploaderError != null && !strFileUploaderError.equalsIgnoreCase(""))
    {
%>
<script language="javascript">
ServerMessage.innerHTML = ServerMessage.innerHTML + "<br/>" +
    "<%=strFileUploaderError%>";
</script>

```

Next, the Java code attempts to get a connection to E-mail Server. If it does not succeed, an error message is displayed:

```

if (action != null && !action.equals(""))
{
    EspEmailProtocol email = new EspEmailProtocol(new Endpoint(new URI("tcp://" +
    svcHost + ":" + String.valueOf(svcPort))));
    boolean bConnected = false;

```

```

        try
        {
            email.open();
            bConnected = true;
        }
        catch (Exception e)
        {
%>
<script language="javascript">
ServerMessage.innerHTML = "We are sorry, Email Server is unavailable at this time.
    Please try again later.";

```

Tracking States and Submission

If the connection is successful and the Submit button was clicked, the properties and contents of the e-mail are submitted in a request to E-mail Server. The following Java code checks the state of the JSP file after a submission, because the JSP calls itself in response to the user events. This code is very similar to the code that is found in the E-mail Sample, except that additional code has been added to include the attachments in the submission:

```

if (bConnected && action.equals("Submit"))
{
    RequestCreateWebEmailIn espRequest = RequestCreateWebEmailIn.create();
    espRequest.setFirstName(first_name);
    espRequest.setLastName(last_name);
    espRequest.setFromAddress(email_address);
    espRequest.setText(body);
    espRequest.setSubject(subject);
    if (reply_from != null && !reply_from.equalsIgnoreCase(""))
        espRequest.setMailbox(reply_from);

```

The following code checks to see if there are attachments. If attachments exist, they are added to the e-mail message before it is submitted. Note that attachments are handled differently from the other e-mail fields, such as first name and last name:

```

if (attachment1 != null || attachment2 != null)
{
    EmailAttachmentList eal = new EmailAttachmentList();

    if(attachment1 != null && attachment1.length > 0)
    {
        EmailAttachment attach1 = new EmailAttachment();
        attach1.setContent(attachment1);
        attach1.setContentType(contenttype1);
        attach1.setFileName(filename1);
        eal.add(attach1);
    }

```

```

        if(attachment2 != null && attachment2.length > 0)
        {
            EmailAttachment attach2 = new EmailAttachment();
            attach2.setContent(attachment2);
            attach2.setContentType(contenttype2);
            attach2.setFileName(filename2);
            eal.add(attach2);
        }
        espRequest.setAttachments(eal);
    }

    Message msg = email.request(espRequest, 30000);...

```

An appropriate message will be displayed, based on whether the submission was successful:

```

else if (msg instanceof EventCreateWebEmailIn)
{
    EventCreateWebEmailIn eventAck = (EventCreateWebEmailIn) msg;
    id = eventAck.getNewInteractionId();
%>
<script language="javascript">
ServerMessage.innerHTML = "Request (<%=mask_html(id)%>) has been successfully submitted
    to Email Server.";
</script>
<%
}
else if (msg instanceof EventError)
{
    EventError eventError = (EventError) msg;
%>
<script language="javascript">
ServerMessage.innerHTML = "Could not submit the email to Email Server.\r\nReason:
    <%=mask_html(eventError.getFaultString())%>";
</script>

```

Closing the Connection

The connection to E-mail Server must be closed. Otherwise, you will have orphan connections to the server that do not relinquish resources— principally, network resources (such as sockets) and memory.

```

if (email != null && bConnected)
    email.close();
}

```

Handling User Events

Four JavaScript functions handle user events:

- `window_onload()`—Sets the request ID for this form. If the document is not undefined, it sets the value of the form ID to the value of the request ID.
- `window_onunload()`—Has no code. It is an empty function.
- `Submit_onClick()`—Sets the action value to `Submit`. Because this is a real submission from the user and not a JSP workaround, the function must verify that the e-mail address and sender data are in an acceptable format. If the function finds invalid characters, it presents a dialog box and requests that the user correct the problem. When the data is acceptable, the function calls the `HTML submit()` function.
- `Reset_onClick()`—Calls the `reset()` method of the form to clear out all of the data that is entered.

Chat Sample

The `.../SimpleSamples802` directory contains files for the Chat, the Advanced Chat, and the Chat and Cobrowse Samples.

This section outlines the purpose, functionality, code implementation, and customization options of the Chat Sample.

Purpose

The Chat Sample demonstrates how to add a simple chat feature to any web form that supports Java libraries.

Functionality Overview

The following sections review the code that is used in the implementation of the different chat functions:

- Users Interface Implementation:
 - “Declaring Variables” on [page 122](#)
 - “Constructing the HTML Body” on [page 122](#)
 - “Handling User Events” on [page 123](#)
- Logic Implementation:
 - “Loading Parameters” on [page 124](#)
 - “Connecting to Chat Server” on [page 125](#)
 - “Creating a Chat Session” on [page 126](#)
 - “Handling User Requests” on [page 127](#)
 - “Masking Data” on [page 129](#)
 - “Processing Server Events” on [page 130](#)
 - “Tracking State” on [page 131](#)
 - “Closing the Connection” on [page 132](#)
 - “Handling User Events” on [page 132](#)

For information about code that is common to all of the samples—such as retrieving parameters, setting the content type, character encoding, loading libraries, and importing files—see “Common Code” on [page 107](#).

Files

The `.../Chat` directory contains the HTML Chat Sample. The sample consists of three files:

- `HtmlChatCommand.jsp`—A file that contains most of the chat logic in a hidden frame.
- `HtmlChatFrameSet.jsp`—A frameset that holds the `HtmlChatCommand.jsp` and `HtmlChatPanel.jsp` files.
- `HtmlChatPanel.jsp`—A file that contains the code that is used to create a chat panel that has input boxes for entering the chat message. All of the data is sent to the parent form.

Code Explanation

The Chat Sample separates user interface (UI) and logic components. The following subsections explain the specific functions and the code for each of these components.

User Interface Implementation

The UI code demonstrates how to present form controls such as panels, input boxes, buttons, and so on. The code resides in the `HtmlChatPanel.jsp` file.

Declaring Variables

The following variables are declared:

```
var SessionID = "";
var bCommandFrameReady = false;
var bConnected = false;
```

Constructing the HTML Body

The HTML code creates input boxes in which clients can fill in their personal information and their message. The code also creates submit buttons so that clients can send their new chat messages:

```
<form name="chat_form" id=chat_form onSubmit="javascript:on_send(); return false;">
  <table style="width:100%" cellpadding="2">
    <tr>
      <td class="samplesSampleTopToolbar">
        <table >
          <tr >
```

```

        <td style="width:435px" class="samplesSampleNameTitleWhite">Chat
sample</td>
        <td style="width:91px" onclick="javascript:on_connect();"
class="sampleGreenButtonBig" onmouseover="javascript:MouseOver(this);"
onmouseout="javascript:MouseOut(this);">Start</td>
        <td style="width:91px" onclick="javascript:on_disconnect();"
class="sampleRedButtonBig" onmouseover="javascript:MouseOver(this);"
onmouseout="javascript:MouseOut(this);">Stop</td>
        <td style="width:auto" ></td>
    </tr>
</table>
</td>
</tr>
<tr>
    <td class="samplesSampleParagraph">Fill in your personal information in
fields below to proceed.</td>
</tr>
<tr>
    <td>
        <table style="width:617px" border="0">
            <tr>
                <td class="samplesSampleFieldTitle" ></td>
                <td></td>
                <td style="width:6px; vertical-align: middle;"></td>
            </tr>...
```

Handling User Events

The body of the `HtmlChatPanel.jsp` file begins with a call to this function:

- `window_onload()`—Sets the logic when the window is first loaded into memory.

These `HtmlChatPanel.jsp` functions currently have no implementation:

- `window_onunload()`
- `clear_transcript()`

The `HtmlChatCommand.jsp` file resides in the frame that is named `itf`. You can invoke these functions from that frame:

- `on_connect()`—Calls the `on_connect()` function in the page that resides in the `itf` frame.
- `on_disconnect()`—Calls the `on_disconnect()` function in the page that resides in the `itf` frame.
- `on_send()`—Calls the `on_send()` function in the page that resides in the `itf` frame. It also prevents new requests from being sent to Chat Server until the response from the previous request has arrived.

- `message_onkeypress()`—The event processor for “user is typing” notification. It sends a notification when the user starts to type into the “message” field.
- `CommandFrameReady()`—Boolean flag for making the frame ready to accept a command.
- `AddMessage()`—Appends new text to the chat transcript.
- `show_message()`—Appends new text to the chat transcript.
- `SetSessionID()`—Sets the SessionID for future use.
- `connected()`—Sets the `bConnected` boolean flag to true and enables the form.
- `disconnected()`—Sets the `bConnected` boolean flag to false and disables the form.

Logic Implementation

The `HtmlChatCommand.jsp` contains the main logic for the functionality of the Chat Sample. The following subsections explain the code that is in the file.

Loading Parameters

The `HtmlChatCommand.jsp` code loads the hidden parameters upon each JSP refresh. On the initial JSP launch, the hidden parameters have no values. After the initial launch, the server returns values and they are stored in the hidden parameters. See the “Tracking State” on [page 131](#) topic for more information:

```
// server side processing
String cmd = i18nsupport.GetSubmitParametr(request, "cmd");

String chat_alias = i18nsupport.GetSubmitParametr(request, "chat_alias");
if( chat_alias == null ) chat_alias = "";

String first_name = i18nsupport.GetSubmitParametr(request, "first_name");
if( first_name == null ) first_name = "";

String last_name = i18nsupport.GetSubmitParametr(request, "last_name");
if( last_name == null ) last_name = "";

String email_address = i18nsupport.GetSubmitParametr(request, "email_address");
if( email_address == null ) email_address = "";

// We maintain secureKey and userId
String secure_key = i18nsupport.GetSubmitParametr(request, "secure_key");
if( secure_key == null ) secure_key = "";

String user_id = i18nsupport.GetSubmitParametr(request, "user_id");
if( user_id == null ) user_id = "...";
```

The `cmd` variable reflects the action or button that the user clicked. If the user clicked the Connect button, the code attempts to get a handle to the Load Balancer:

```
if( cmd != null )
{
    String svcHost = null;
    int svcPort = -1; ...
```

Connecting to Chat Server

Chat Server works differently from E-mail Server, whose behavior is described in “E-Mail Sample” on [page 108](#). The key difference is that Chat Server must continually connect, check the transcript (send/receive messages), and disconnect. (In e-mail, clients fill out a form. After they submit the form, the transaction is over. Chat Server, however, must maintain the identity of each client until one party ends the session).

The following `HtmlChatCommand.jsp` code snippet shows what happens after the sample code has created an instance of Load Balancer and has tried to connect to Chat Server. At this point, the JSP file checks the user-name values. If those values are not available, the connection cannot continue:

```
//Create load balancer instance
if( cmd.equals("connect") )
{
    //User wants to establish a chat session, at this point, select chat server. Using
    //load balancer and use this server for the rest of the session.

    if( first_name.equals("") || last_name.equals("") )
    {
        itf_response = "USERNAMEREQUIRED";
        itf_message = "Please enter first and last names.";
    }
    else
    {
        ServiceInfo si = LoadBalancer.GetServiceInfo
            (CfgAppType.CFGChatServer, strTenant);
        svcHost = si.getHost();
        svcPort = si.getWebApiPort();
        if(svcHost == null || svcHost.equalsIgnoreCase("")) ||
            svcPort == -1)
        {
            itf_response = "NOSERVICE";
            itf_message = "Chat service is unavailable at this time, please try again later.";
        } ...
    }
}
```

If the Load Balancer finds a Chat Server, the code returns an alias to that Chat Server, and then tries to connect and login the user. The alias is saved and will be used in the future to connect to the same chat server each time:

```

else
{
// we have a chat server at this point
chat_alias = si.getAlias();...

```

Creating a Chat Session

Now, you must send your user information and create a login request.

```

try
{
    FlexChatProtocol chat = new FlexChatProtocol (new
Endpoint(new URI("tcp://" + svcHost + ":" + String.valueOf(svcPort))));
    chat.setAutoRegister(false);
    chat.open();

    KeyValueCollection userdata = new KeyValueCollection();
    userdata.addString(fldnFirstName, first_name);
    userdata.addString(fldnLastName, last_name);
    userdata.addString(fldnEmailAddress, email_address);
    userdata.addString("IdentifyCreateContact", "3"); //1=No
identification, 2=Identification, no creation, 3=Identification and creation
(default value)

    userdata.addString("MediaType", "chat");

    String strNickName = "";
    if (last_name != null && last_name.length() > 0)
        strNickName = first_name + last_name.charAt(0);
    else
        strNickName = first_name;
    RequestLogin rl = RequestLogin.create(strNickName, new
Integer(8), userdata);
    rl.setTimezoneOffset(Integer.parseInt(timeZoneOffset));

    msg = chat.request(rl, 30000);

```

If Chat Server responds with a **CONNECTED** status, the code displays a welcome message. Otherwise, it displays an error message:

```

if (msg != null)
{
    if (msg.messageId() == EventStatus.ID)
    {
        EventStatus status = (EventStatus)msg;
        if (status.getReferenceId().intValue() != 0 && status.getSecureKey() != null)
        {
            secure_key = status.getSecureKey();
            user_id = status.getUserId();
            itf_response = "CONNECTED";
            itf_message = "Welcome to Genesys chat!";...

```

Next, the code makes a request to join a chat session. If Chat Server returns a `CONNECTED` status, the user joins the new session successfully. Otherwise, the JSP returns an error description that is received from Chat Server:

```
RequestJoin rj = RequestJoin.create(user_id, secure_key, "", strTenant + ":" +
    strQueueKey, subject);
msg = chat.request(rj, 30000);
EventStatus es = (EventStatus) msg;
if (es != null && es.getRequestResult() == RequestResult.Success)
{
    clear_transcript = true;
    transcript = es.getFlexTranscript();
    script_pos = transcript.getLastPosition().toString();
    itf_response = "CONNECTED";
}
else
{
    itf_response = "DISCONNECTED";
    if (es.getDescription() != null)
        itf_message = es.getDescription().getText();
    else
        itf_message = "Could not create chat session.";
}
}
else
{
    itf_response = "DISCONNECTED";
    itf_message = "Not connected to chat server, unexpected error.";
}
```

Handling User Requests

This section shows what the JSP does if the user is not requesting a connection to Chat Server. The possible requests are; disconnect from the server, to send a chat message, or the user is typing. The code reacts differently to each request:

```
if (cmd.equals("disconnect"))
{
    RequestLogout rlo = RequestLogout.create(user_id, secure_key,
new Integer(0));
    msg = chat.request(rlo, 30000);

    itf_response = "DISCONNECTED";
    itf_message = "Chat was finished";
}
else if (cmd.equals("send"))
{
    EventStatus es = null;
    try
    {
```

```

        RequestRefresh rr = RequestRefresh.create(user_id,
secure_key, new Integer(Integer.parseInt(script_pos)+1) , MessageText.create("text",
TreatAs.NORMAL, msg2send.equalsIgnoreCase("") ? null : msg2send));
        msg = chat.request(rr, 30000);
        if (msg != null && msg.messageId() == EventStatus.ID)
        {
            es = (EventStatus) msg;
            esCurrentStatus = es;

            transcript = es.getFlexTranscript();
            script_pos = transcript.getLastPosition().toString();
            itf_response = "TRANSCRIPT";
        }
    }
    catch (Exception ex)
    {
        itf_response = "SENDFAILED";
        if (es != null && es.getDescription().getText() != null)
            itf_message = es.getDescription().getText();
        else
            itf_message = "Can't get chat transcript from incoming
packet. " + ex.toString();
    }
}
else if (cmd.equals("user_typing"))
{
    EventStatus es = null;
    try
    {
        RequestRefresh rr = RequestRefresh.create(user_id,
secure_key, new Integer(Integer.parseInt(script_pos) + 1),
NoticeText.create(NoticeType.TypingStarted, ""));
        msg = chat.request(rr, 30000);
        if (msg != null && msg.messageId() == EventStatus.ID)
        {
            es = (EventStatus) msg;
            esCurrentStatus = es;

            transcript = es.getFlexTranscript();
            script_pos = transcript.getLastPosition().toString();
            itf_response = "TRANSCRIPT";
        }
    }
    ...
}

```

The send and user_typing commands are similar except for the last parameter that is passed by the user_typing command. The value of the last parameter should be TreatAs.SYSTEM, which will ensure that the message is not displayed to the client side.

Masking Data

The `MaskSymbols()` method masks the symbols that are dangerous to the client browser. For example, it translates `<script>` to the harmless `"<" + "script" + ">"`. The script that is inside of these tags will not be executed:

```
public String MaskSymbols (String strIn)
{
    String strOut = "";
    int i;
    int iLen;
    char ch;
    if (strIn != null)
    {
        iLen = strIn.length();
        for (i=0; i < iLen; i++)
        {
            ch = strIn.charAt(i);
            if (ch == '\r')
                strOut += "\\r";
            else if (ch == '\n')
                strOut += "\\n";
            else if (ch == '\t')
                strOut += "\\t";
            else if (ch == '"')
                strOut += "\\\"";
            else if (ch == '\\')
                strOut += "\\\\";
            else if (ch == '<')
            {
                if (i != iLen-1)
                    strOut += "\" + \"<\" + \"";
                else
                    strOut += "\" + \"<\"";
            }
            else if (ch == '>')
            {
                if (i != iLen-1)
                    strOut += "\" + \">\" + \"";
                else
                    strOut += "\" + \">\"";
            }
            else
                strOut += ch;
        }
    }
    return strOut; ...
}
```

Processing Server Events

When the sample receives a server event, the code calls the `getEventType()` method to check the current status of the chat packet. In the following code snippet, the code checks for connection, abandonment, and message flags:

```
if( transcript != null )
{
    out.write("parent.main.SetSessionID(\"" + transcript.getSessionId() + "\");\r\n");
    EventInfoList eil = transcript.getEventInfoList();
    if (eil != null)
    {
        for(int i = 0; i < eil.size(); i++ )
        {
            EventInfo ei = eil.getPartyInfo(i);
            String text2append = "";
            if (ei.getEventType() == EventType.Connect)
            {
```

You must hide `SYSTEM` messages from server or treat them individually:

```
if (ei.getUserType() != UserType.External)
{
    text2append = text2append + "New party ('" + ei.getUserNickname()
+ "') has joined the session";
    if (ei.getText() != null && !ei.getText().equalsIgnoreCase(""))
        text2append = text2append + ": " + ei.getText();
}
else if (ei.getEventType() == EventType.Message)
{
    if (ei.getText() != null && !ei.getText().equalsIgnoreCase(""))
        text2append = text2append + ei.getUserNickname() + ": " +
ei.getText();
    else
        text2append = text2append + ei.getUserNickname() + ":";
}
else if (ei.getEventType() == EventType.Abandon)
{
    text2append = text2append + "Party ('" + ei.getUserNickname() + "')
has left the session.";
}
else if (ei.getEventType() == EventType.Notice)
{
    if (esCurrentStatus != null &&
esCurrentStatus.getPartyId().intValue() != ei.getPartyId().intValue())
    {
        if (ei.getNoticeType() == NoticeType.TypingStarted)
        {
```

```

        text2append = text2append + ei.getUserNickname() + " is
typing.";
    }
    else if (ei.getNoticeType() == NoticeType.TypingStopped)
    {
        text2append = text2append + ei.getUserNickname() + " has
stopped typing.";
    }
    else if (ei.getNoticeType() == NoticeType.PushUrl)
    {
        text2append = text2append + "Please navigate to the link:
"+MaskSymbols(ei.getText());
    }
}

if (!text2append.equals(""))
    out.write ("parent.main.show_message(\""+MaskSymbols(text2append) +
"\");");

```

Tracking State

The following sample has many hidden variables to help track the state of the JSP file:

```

<form method="post" action="HtmlChatCommand.jsp">
<input type="hidden" id="cmd" name="cmd">
<input type="hidden" id="chat_alias" name="chat_alias"
    value="<%=mask_html(chat_alias)%>">
<input type="hidden" id="first_name" name="first_name"
    value="<%=mask_html(first_name)%>">
<input type="hidden" id="last_name" name="last_name" value="<%=mask_html(last_name)%>">
<input type="hidden" id="email_address" name="email_address"
    value="<%=mask_html(email_address)%>">
<input type="hidden" id="subject" name="subject" value="<%=mask_html(subject)%>">
<input type="hidden" id="msg2send" name="msg2send">
<input type="hidden" id="secure_key" name="secure_key"
    value="<%=mask_html(secure_key)%>">
<input type="hidden" id="user_id" name="user_id" value="<%=mask_html(user_id)%>">
<input type="hidden" id="script_pos" name="script_pos"
    value="<%=mask_html(script_pos)%>">
<input type="hidden" id="session_id" name="session_id"
    value="<%=mask_html(session_id)%>">
<input type="hidden" id="timeZoneOffset" name="timeZoneOffset"
    value="<%=mask_html(timeZoneOffset)%>">
</form>

```

Some of these hidden variables are straightforward, while others require some explanation:

- `chat_alias` is the name of the Chat Server.
- `secure_key` is a security measure for data transmission.
- `script_pos` refers to the position of a transcript character.

Closing the Connection

When the user is ready to submit the form, your client application must close the connection to the Chat Server and reset the service dispatcher to `null`. Otherwise, there will be orphan connections to the server that do not relinquish their resources—principally, network resources (such as sockets) and memory:

```
if( chat != null )
{
    chat.close();
    chat = null;
}
```

In addition, before ending a chat by logging out of Chat Server, the code of your client application must ensure that the application waits to receive a reply from Chat Server to the Connect request of the browser. Otherwise, the client who is logged out will leave behind a pending interaction that Genesys Desktop will be unable to delete.

Handling User Events

Six JavaScript functions handle user events:

- `window_onload()`—Calls only the disconnected method that sets the required variables and form elements to the appropriate state.
- `on_connect()`—Sets the `cmd` variable to `connect`, sets the necessary data to connect to Chat Server, and calls the HTML form `submit()` function.
- `on_disconnect()`—Sets the `cmd` variable to `disconnect` and calls the HTML form `submit()` function.
- `on_send()`—Sets the `cmd` variable to `send`, sets the message that is to be send by using the value from the `strMessage` argument, and then the HTML form `submit()` function.
- `on_refresh()`—Sets the `cmd` variable to `send` and calls the HTML form `submit()` function.
- `on_user_typing()`—Displays a system message on the desktop of the agent, but does not display it on the screen of the user.

Advanced Chat

This section presents the purpose, functionality overview, and code implementation of the Advanced Chat Sample.

Purpose

The Advanced Chat Sample demonstrates how to add a chat feature to any web form. It also supports smiles and hyperlinks.

Functionality Overview

The Advanced Chat Sample includes only one feature that is different from the previously described Chat Sample. The following sections review the code that is used to implement the supported smiles and hyperlink functionality. For a complete overview of the Advanced Chat functions, see Chapter 9, “Chat Sample,” on [page 121](#):

- “Handling User Events” on [page 133](#)
- “Including Emoticons and Hyperlinks” on [page 134](#)

For information about code that is common to all of the samples— such as retrieving parameters, setting the content type, character encoding, loading libraries, and importing files—see “Common Code” on [page 107](#).

Files

The `.../AdvancedChat` directory contains the Advanced Chat Sample. The sample consists of four files:

- `HtmlChatCommand.jsp`—A file that contains most of the chat logic.
- `HtmlChatFrameSet.jsp`—A frameset that holds the `HtmlChatCommand.jsp` and `HtmlChatPanel.jsp` files.
- `HtmlChatPanel.jsp`—A file that contains the code that is used to create a chat panel that has input boxes in which to enter the chat message. All of the data is sent to the parent form.
- `ChatTranscript.jsp`—A file that supports popular Internet expressions— such as smiling or frowning facial expressions and hyperlinks—in an e-mail or chat communication.

Code Explanation

The Advanced Chat Sample separates UI and logic components in the same manner as the Chat Sample. The Advanced Chat and Chat Sample code is similar, except for a few minor changes. The main difference is the implementation of emotion icons and hyperlinks.

Handling User Events

The functionality for emotion icons and hyperlinks is contained within the four methods of the `ChatTranscript.jsp` file:

- `AddMessage()`—Adds messages from the server to the chat-transcript panel.

- `ParseSmilesAndLinks()`—Parses message that are to be displayed. This method shows links as clickable hyperlinks and smiles as emoticons.
- `ProcessOneWord()`—Checks each word in the message to determine if it is an emoticon or a hyperlink.
- `HideHTML()`—Prevents hacking by hiding all potentially dangerous content, such as the following symbols: `< " > &`. This functionality makes the incoming message harmless to the client.

Including Emoticons and Hyperlinks

An array of GIF files that are found in the `ChatTranscript.jsp` file are used for emotion icons. You can replace any of these GIF files with your own to customize your chat box. In order for the text to be recognized as a hyperlink it must begin with one of the following formats:

- `http:\\`
- `http://`
- `https:\\`
- `https://`
- `www.`

During a chat, the user clicks an icon that represents a facial expression. The `ProcessOneWord()` function determines that an emoticon is present and then the `ParseSmilesAndLinks()` function parses the message. When the user clicks to send the message, the correct image is inserted in the message box.

Chat Widget Sample

This section presents the purpose, functionality overview, and code implementation of the Chat Widget Sample.

Purpose

The Chat Widget Sample demonstrates how to customize a chat window.

Functionality Overview

The Chat Widget Sample includes files that contain the code for customizing your chat window. The following sections review the code that is used in the implementation of the different customization functions:

- “Collecting Parameters” on [page 135](#)
- “Displaying a Customized Window” on [page 137](#)
- “Handling User Events” on [page 137](#)

For information about code that is common to all of the samples—such as retrieving parameters, setting the content type, character encoding, loading libraries, and importing files—see “Common Code” on [page 107](#).

Files

The `.../ChatWidget` directory contains the Chat Widget Sample. The sample consists of five files:

- `HtmlChatCommand.jsp`—Controls the behavior of the chat page, based on user commands
- `HtmlChatFrameSet.jsp`—Contains the frameset for the other two chat files
- `HtmlChatPanel.jsp`—Controls the display in the chat panel
- `ChatTranscript.jsp`—Supports popular Internet expressions such as smiling or frowning facial expressions and hyperlinks in an e-mail or chat communication
- `ChatWidget.html`—Contains the main form where you can customize the chat window.

Code Explanation

The following subsections explain the code in the `HtmlChatFrameSet.html` and `ChatTranscript.jsp` files. The code examples demonstrate how parameters can be collected from a top-level page and then passed to sub-frames.

Collecting Parameters

The `HtmlChatFrameSet.html` file contains the code needed to collect parameters and then pass the parameters to a sub-frame:

```
<%
String strActionColor      = "FF00FF";
String strClientNickNameColor = "0000CD";
String strAgentNickNameColor = "008000";
String strClientMessageColor = "1532a0";
String strAgentMessageColor  = "708090";
String strFontName         = "arial";
String strFontSize          = "3";
String strBackgroundColor    = "FFFFFF";
String strShowSmiles        = "1";
String strLogURL             = "../Resource/Images/logo.jpg";
String URI                  = "";

String strTmp = "";

strTmp = i18nsupport.GetSubmitParametr(request, "ActionColor");
if (strTmp != null && strTmp.compareToIgnoreCase("") != 0)
    strActionColor = strTmp;
```

```

strTmp = i18nsupport.GetSubmitParametr(request, "ClientNickNameColor");
if (strTmp != null && strTmp.compareToIgnoreCase("") != 0)
    strClientNickNameColor = strTmp;

strTmp = i18nsupport.GetSubmitParametr(request, "AgentNickNameColor");
if (strTmp != null && strTmp.compareToIgnoreCase("") != 0)
    strAgentNickNameColor = strTmp;

strTmp = i18nsupport.GetSubmitParametr(request, "ClientMessageColor");
if (strTmp != null && strTmp.compareToIgnoreCase("") != 0)
    strClientMessageColor = strTmp;

strTmp = i18nsupport.GetSubmitParametr(request, "AgentMessageColor");
if (strTmp != null && strTmp.compareToIgnoreCase("") != 0)
    strAgentMessageColor = strTmp;

strTmp = i18nsupport.GetSubmitParametr(request, "Logo");
if (strTmp != null && strTmp.compareToIgnoreCase("") != 0)
    strLogURL = strTmp;

strTmp = i18nsupport.GetSubmitParametr(request, "FontName");
if (strTmp != null && strTmp.compareToIgnoreCase("") != 0)
    strFontName = strTmp;

strTmp = i18nsupport.GetSubmitParametr(request, "FontSize");
if (strTmp != null && strTmp.compareToIgnoreCase("") != 0)
    strFontSize = strTmp;

strTmp = i18nsupport.GetSubmitParametr(request, "BackgroundColor");
if (strTmp != null && strTmp.compareToIgnoreCase("") != 0)
    strBackgroundColor = strTmp;

strTmp = i18nsupport.GetSubmitParametr(request, "ShowSmiles");
if (strTmp != null && strTmp.compareToIgnoreCase("") != 0)
    strShowSmiles = strTmp;

URI = "HtmlChatFrameSet.jsp?";
URI += "ActionColor=" + URLEncoder.encode(strActionColor);
URI += "&ClientNickNameColor=" + URLEncoder.encode(strClientNickNameColor);
URI += "&AgentNickNameColor=" + URLEncoder.encode(strAgentNickNameColor);
URI += "&ClientMessageColor=" + URLEncoder.encode(strClientMessageColor);
URI += "&AgentMessageColor=" + URLEncoder.encode(strAgentMessageColor);
URI += "&Logo=" + URLEncoder.encode(strLogURL);
URI += "&FontSize=" + URLEncoder.encode(strFontSize);
URI += "&FontName=" + URLEncoder.encode(strFontName);
URI += "&ShowSmiles=" + URLEncoder.encode(strShowSmiles);
URI += "&BackgroundColor=" + URLEncoder.encode(strBackgroundColor);
%>

```

Displaying a Customized Window

The parameters that were collected and passed to the `ChatTranscript.jsp` file are now used to change the font name, font size, and background color of the transcript. Customize the window further by including a company logo, for instance. You can extend this functionality by including your specific parameters:

```
<%
String strFontName = "arial";
String strFontSize = "3";
String strBackgroundColor = "FFFFFF";
String strTmp = "";

strTmp = i18nsupport.GetSubmitParametr(request, "FontName");
if (strTmp != null && strTmp.compareToIgnoreCase("") != 0)
    strFontName = strTmp;

strTmp = i18nsupport.GetSubmitParametr(request, "FontSize");
if (strTmp != null && strTmp.compareToIgnoreCase("") != 0)
    strFontSize = strTmp;

strTmp = i18nsupport.GetSubmitParametr(request, "BackgroundColor");
if (strTmp != null && strTmp.compareToIgnoreCase("") != 0)
    strBackgroundColor = strTmp;
%>...

...<body style="background-color1:#{<%=strBackgroundColor%}>">
<script type="text/javascript" language="javascript">
var FontName = "<%=strFontName%>";
var FontSize = "<%=strFontSize%>";
var BackgroundColor = "<%=strBackgroundColor%>";
var TimestampColor = "000000";...
```

Handling User Events

The following JavaScript functions handle user events:

- `window_onload()`—Called every time that the page is loaded. Sets the logic when the window is first loaded into memory.
- `window_onunload()` has no implementation.

The `window_onresize()` function helps to resize the form's elements dynamically. The fields in the window maintain their size and shape even when the user resizes the window:

```
if (iWidth > 380)
{
    if (navigator.userAgent.toLowerCase().indexOf("firefox") != -1)
    {
        document.forms[0].FirstName.style.width = (iWidth - 310) + "px";
```

```

        document.forms[0].LastName.style.width = (iWidth - 310) + "px";
        document.forms[0].EmailAddress.style.width = (iWidth - 310) + "px";
        document.forms[0].Subject.style.width = (iWidth - 310) + "px";
        document.forms[0].message.style.width = (iWidth - 284) + "px";
        document.getElementById("ChatTranscript").style.width = (iWidth - 15) +
"px";
    }
    else
    {
        document.forms[0].FirstName.style.width = (iWidth - 270) + "px";
        document.forms[0].LastName.style.width = (iWidth - 270) + "px";
        document.forms[0].EmailAddress.style.width = (iWidth - 270) + "px";
        document.forms[0].Subject.style.width = (iWidth - 270) + "px";
        document.forms[0].message.style.width = (iWidth - 253) + "px";
        document.getElementById("ChatTranscript").style.width = (iWidth - 15) +
"px";
    }
}

```

Survey Sample

This section presents the purpose, functionality overview, and code implementation of the Survey Sample. This survey sample is included in the Advanced Chat Sample. Some references to the Advanced Chat Sample will be made to clarify the code that enables the survey to be displayed.

Purpose

The Survey Sample demonstrates how to include a survey in a chat window.

Functionality Overview

The following sections review the code that is used in the implementation of the different survey functions:

- “Displaying the Survey” on [page 139](#)
- “Gathering the Survey Results” on [page 140](#)
- “Submitting the Survey Results” on [page 141](#)

For information about code that is common to all of the samples—such as retrieving parameters, setting the content type, character encoding, loading libraries, and importing files—see “Common Code” on [page 107](#).

Files

The `.../Survey` directory contains the Survey Sample. The sample consists of one file:

- `Survey.jsp`—Contains code snippets that use the survey portion of the API to initialize a survey.

Code Explanation

The following subsections explain the code in the `AdvancedChat/HtmlChatPanel.jsp` file and the `Survey.jsp` file.

Displaying the Survey

The Advance Chat Sample displays a checkbox associated with survey on the chat window. When the box is checked, the survey will be presented to the customer once the chat session has been completed. The following code appears in the `AdvancedChat/HtmlChatPanel.jsp` file to display the survey check box:

```
...Take survey after chat <input type="checkbox" id="SurveyAfterChat"
name="SurveyAfterChat" checked="checked" />...
```

The file also contains a user event handler and two functions associated with the survey functionality:

The event handler, `TranscriptAfterChat_onclick()`, determines if the check box is selected:

```
function TranscriptAfterChat_onclick()
{
    if (bConnected == true)
    {
        bCommandFrameReady = false;
        parent.itf.on_send_transcript(document.forms[0].SurveyAfterChat.checked);
    }
}
```

The `GetSurveyUrl()` function returns the survey's URL:

```
function GetSurveyUrl()
{
    var theURL = unescape(location.href);
    var iPos = theURL.indexOf("SimpleSamples802", 0);
    if (iPos != -1)
    {
        theURL=theURL.substring(0, iPos);
        theURL = theURL + "SimpleSamples802/" + "Survey/Survey.jsp?ParentID="+SessionID;
    }
}
```

```

else
{
    theURL ="http://localhost/SimpleSamples802/Survey/Survey.jsp?ParentID="+SessionID;
}
return theURL;
}

```

The function, `disconnected()` calls `GetSurveyUrl()` and opens the survey in a new window. It also determines if a pop-blocker is being used and sends a message to suggest that the customer disable the blocker:

```

function disconnected()
{
    bConnected = false;
    document.forms[0].send.disabled = true;
    if (document.forms[0].SurveyAfterChat.checked && SessionID != "")
    {
        var strMessage = GetSurveyUrl(); //"../Survey/Survey.aspx?ParentID="+SessionID;
        var bNeedToShowMessage = false;
        var winNewWindow = window.open (strMessage, "survey", "", false);

        if (winNewWindow == null)
            bNeedToShowMessage = true;
        else
        {
            if (window.opera)
            {
                if (!winNewWindow.opera)
                    bNeedToShowMessage = true;
            }
        }

        if (bNeedToShowMessage)
        {
            window.frames.ChatTranscript.AddMessage ("System: pop-up blocker detected. ",
                AgentNickNameColor, 1, 1);
            window.frames.ChatTranscript.AddMessage ("Please navigate to the next link to
                take our survey: " + strMessage, AgentMessageColor, 0, 0);
        }
    }
}...

```

Gathering the Survey Results

The `Survey.jsp` file contains the code to create the survey form and the event handlers and functions needed to gather the customer's selections and submit the results.

The `setCheckedValue(radioObj, newValue)` function allows you to change the values of the radio buttons on the form:

```
function setCheckedValue(radioObj, newValue)
{
    if(!radioObj)
        return;

    var radioLength = radioObj.length;

    if(radioLength == undefined)
    {
        radioObj.checked = (radioObj.value == newValue.toString());
        return;
    }

    for(var i = 0; i < radioLength; i++)
    {
        radioObj[i].checked = false;
        if(radioObj[i].value == newValue.toString())
            radioObj[i].checked = true;
    }
}
```

The `window_onload()` event handler is called every time that the page is loaded. This function calls the `setCheckedValue(radioObj, newValue)` function and clears any selected radio buttons on the form when it is initial opened:

```
function window_onload()
{
    if ("<%=strExperienceAnswer%>" != "")
        setCheckedValue (document.forms[0].elements["RadioBtnList_Experience"],
            "<%=strExperienceAnswer%>");
    if ("<%=strResolutionAnswer%>" != "")
        setCheckedValue (document.forms[0].elements["RadioBtnList_Resolution"],
            "<%=strResolutionAnswer%>");
    if ("<%=strRecommendAnswer%>" != "")
        setCheckedValue (document.forms[0].elements["RadioBtnList_Recommend"],
            "<%=strRecommendAnswer%>");
}
```

Submitting the Survey Results

This section of code creates an instance of Load Balancer and an instance of Interaction Server by using the host and port IDs of Interaction Server. If there are any errors or exceptions, the code handles them:

```
try
{
    ServiceInfo si = LoadBalancer.GetServiceInfo(CfgAppType.CFGInteractionServer,
        strTenant);
    strInteractionServerHost = si.getHost();
    iInteractionServerPort = si.getPort();
}
```

```

catch (LoadBalancerException e)
{
    bError = true;
    strMessage = "We are sorry, Interaction Server is unavailable at this time. Please
        try it again later or enter information manually.";
}
try
{
    if (strInteractionServerHost != null &&
        !strInteractionServerHost.equalsIgnoreCase("") && iInteractionServerPort != -1 &&
        bError == false)
    {
        itxProtocol = new InteractionServerProtocol(new Endpoint(new URI("tcp://" +
            strInteractionServerHost + ":" + String.valueOf(iInteractionServerPort))));
        itxProtocol.setClientType(InteractionClient.MediaServer);
        itxProtocol.setClientName("MCR_WebAPIServer8021");

        try
        {
            itxProtocol.open();
            bConnected = true; ...

```

Once connected, retrieve the tenantID:

```

if (bConnected && bError == false)
{
    int iTenantId = -1;
    if (strTenantID == null || strTenantID.equalsIgnoreCase("") == true)
        strTenantID = LoadBalancer.getTenantId(strTenant);
    try
    {
        iTenantId = Integer.parseInt(strTenantID);
    }
    catch (Exception e1)
    {
        bError = true;
        strMessage = "Can't get tenant ID for " + strTenantID + " tenant. Please check your
            configuration."; ...

```

Create the reqSubmit object:

```

...if (bError == false)
{
    RequestSubmit reqSubmit = RequestSubmit.create();

```

A collection of key-value pairs containing the survey questions and answers. For example, the key for a question is `survey_question_FIELDNAME`. The answer to this question is stored in `survey_answer_FIELDNAME` key. The order number of the question is stored in `survey_question_number_FIELDNAME` key.

The prefixes `survey_question_`, `survey_answer_` and `survey_question_number_` are hardcoded and allow us to separate survey related data from other information in the attached data. When processing the user data you may match the original question with a correct answer by the common postfix `FIELDNAME`.

```
KeyValueCollection kvcUserData = new KeyValueCollection();

kvcUserData.addString(strSurveyQuestionPrefix + "Experience", "How would you rate
    your overall experience");
kvcUserData.addString(strSurveyAnswerPrefix + "Experience", strExperienceAnswer);
kvcUserData.addString(strSurveyQuestionNumberPrefix + "Experience", "0");

kvcUserData.addString(strSurveyQuestionPrefix + "Resolution", "How would you rate the
    resolution we provided");
kvcUserData.addString(strSurveyAnswerPrefix + "Resolution", strResolutionAnswer);
kvcUserData.addString(strSurveyQuestionNumberPrefix + "Resolution", "1");

kvcUserData.addString(strSurveyQuestionPrefix + "Recommend", "Would you recommend us
    to your friends or family");
kvcUserData.addString(strSurveyAnswerPrefix + "Recommend", strRecommendAnswer);
kvcUserData.addString(strSurveyQuestionNumberPrefix + "Recommend", "2");

kvcUserData.addString(strSurveyQuestionPrefix + "Text", "Please provide your overall
    Experience with us");
kvcUserData.addString(strSurveyAnswerPrefix + "Text", strCommentsAnswer);
kvcUserData.addString(strSurveyQuestionNumberPrefix + "Text", "3");
```

The `contactID` is obtained and added to the collection of user data information along with the survey results:

```
if (strContactID == null || strContactID.equalsIgnoreCase("") == true)
{
    try
    {
        InteractionAttributes ia = GetInteractionAttributesByInteractionId(strParentID,
            strTenant);
        if (ia != null)
        {
            strContactID = ia.getContactId();
        }...
    }...
    if (strContactID != null && strContactID.equalsIgnoreCase("") == false)
    kvcUserData.addString(InteractionAttributeListConstants.CONTACT_ID, strContactID);
    kvcUserData.addString(InteractionAttributeListConstants.SUBJECT,
        strSurveySubject);...
```

The `reqSubmit` attributes are set and then submitted to Interaction Server with `MediaType=survey`:

```

reqSubmit.setUserData(kvcUserData);
reqSubmit.setParentInteractionId(strParentID);
reqSubmit.setMediaType(strSurveyMediatype);
reqSubmit.setQueue(strSurveyQueue);
reqSubmit.setTenantId(iTenantId);
reqSubmit.setInteractionType("Inbound");
reqSubmit.setInteractionSubtype("InboundNew");
reqSubmit.setIsOnline(Boolean.FALSE); ...

```

The appropriate message is displayed to the client:

```

Message msg = itxProtocol.request(reqSubmit, 30000);
if (msg != null && msg instanceof EventAck)
{
    EventAck eventAck = (EventAck) msg;
    strMessage = "Survey successfully submitted to our system with reference number "
        + eventAck.getExtension().getString("InteractionId");
    bDisableForm = true;
}
else if (msg != null && msg instanceof EventError)
{
    bError = true;
    EventError eventError = (EventError) msg;
    strMessage = "Can't submit interaction to the InteractionServer. " +
        eventError.getErrorDescription();
}
itxProtocol.close(); ...

```

The submit_survey() function submits the survey results of the survey:

```

function submit_survey ()
{
    document.forms[0].Action.value = "Submit";
    document.forms[0].submit();
}

```

Web Callback Sample

Purpose

The Web Callback Sample is a JSP file that shows how to:

- Create a web callback interaction
- Display a web callback interaction
- Display a list of web callback interactions
- Cancel a web callback interaction
- Reschedule a web callback interaction

Functionality Overview

The following sections review the code that is used in the implementation of the different callback functions:

- “Getting Load Balancer, Interaction Server, and Creating the Protocol Object” on [page 145](#)
- “Create the Web Callback Object” on [page 146](#)
- “Create a Web Callback Interaction” on [page 147](#)
- “Retrieve a Single Web Callback Interaction” on [page 149](#)
- “Retrieve a List of Web Callback Interactions” on [page 149](#)
- “Cancel a Web Callback Interaction” on [page 150](#)
- “Reschedule a Web Callback Interaction” on [page 151](#)

For information about code that is common to all of the samples—such as retrieving parameters, setting the content type, character encoding, loading libraries, and importing files—see “Common Code” on [page 107](#).

Files

The `.../WebCallback` directory contains the Web Callback Sample. The sample consists of a single file, `WebCallback.jsp`.

Code Explanation

The following subsections explain the code in the `WebCallback.jsp` file. In some places several lines of code have been omitted in order to focus your attention on the more important points. In these cases, the missing lines have been replaced with “...”.

Getting Load Balancer, Interaction Server, and Creating the Protocol Object

This section of code creates an instance of Load Balancer and returns an available instance of Interaction Server. When you get an available Interaction Server, store the information about its alias in a hidden element and continue to use the same server every time:

```
if(isProtocol != null && isProtocol.getState() != ChannelState.Opened)
{
    try
    {
        isProtocol.close();
    }
    catch (Exception e)
    {}
    finally
    {
        isProtocol = null;
    }
}
```

```

    }
}

if(isProtocol == null)
{
    try
    {
        ServiceInfo si = LoadBalancer.GetServiceInfo(CfgAppType.CFGInteractionServer,
            strTenant);
        int tenantId = Integer.parseInt(LoadBalancer.getTenantId(strTenant));
        Below, we create the protocol object, and set the client type and name before
        opening it:

```

```

        isProtocol = new InteractionServerProtocol(new Endpoint(new URI("tcp://" +
            si.getHost() + ":" + si.getPort())));
        isProtocol.setClientType(InteractionClient.AgentApplication);
        isProtocol.setClientName("WebCallback Sample");
        isProtocol.open();

```

Next, we create an agent login request, and set the TenantId and PlaceId:

```

RequestAgentLogin requestAgentLogin = RequestAgentLogin.create();
requestAgentLogin.setTenantId(tenantId);
requestAgentLogin.setPlaceId(wcbPlaceId);

Message respondingEvent = isProtocol.request(requestAgentLogin);
if(respondingEvent.messageId() == EventError.ID)
{
    errorMessage = "Could not login to a place " + wcbPlaceId;
    isProtocol.close();
    isProtocol = null;
}
}
catch (Exception e)
{
    isProtocol = null;
    errorMessage = "Error could not connect to Interaction Server. Reason : " +
        e.getMessage();...

```

Create the Web Callback Object

Then we create the WebCallback object, and set the TenantId and various queues:

```

if(wcProtocol == null)
{
    try
    {
        wcProtocol = new WebCallback();

```

```

        wcProtocol.enableLogging(LoadBalancer.getRootLogger());

        wcProtocol.setTenantId(Integer.parseInt(LoadBalancer.getTenantId(strTenant)));
        wcProtocol.setNewQueue(wcbNewQueue);
        wcProtocol.setCancelQueue(wcbCancelQueue);
        wcProtocol.setConferenceQueue(wcbConferenceQueue);
        wcProtocol.setTransferQueue(wcbTransferQueue);
        wcProtocol.setRescheduleQueue(wcbRescheduleQueue);
    }
    catch (Exception e)
    {
        wcProtocol = null;
        errorMessage = "Error could not initialize WebCallback protocol. Reason : " +
            e.getMessage(); ...
    }

```

Create a Web Callback Interaction

The following code creates a web callback interaction using the WCBRequestSubmit method:

```

if (fsAction.Action.getValue() == Actions.SubmitRequestCallback)
{
    if (fsMain.FirstName.getValue().equals("") || fsMain.LastName.getValue().equals("")
        || fsMain.EmailAddress.getValue().equals("") ||
        fsMain.PhoneNumber.getValue().equals("") || fsMain.Subject.getValue().equals(""))
    {
        printErrorMessage(out, "Missing mandatory values");
    }
    else if (fsMain.PhoneNumber.getValue().contains("") ||
        fsMain.PhoneNumber.getValue().contains("\\"))
    {
        printErrorMessage(out, "Phone number (or IP) contains invalid characters");
    }
    else
    {
        WCBRequestSubmit wcbRequest = WCBRequestSubmit.create(fsMain.PhoneNumber.getValue());
        wcbRequest.setSubject(fsMain.Subject.getValue());
        wcbRequest.setDesiredResponseType(fsMain.Media.getValue()); ...
    }
}

```

Schedule a time for the web callback.

```

if (fsMain.CallbackTime.getValue() == 0)
{
    wcbRequest.setType(WebCallbackType.ASAP);
}
else
{
    wcbRequest.setType(WebCallbackType.Scheduled);
}

```

```

TimeZone UTCTimeZone = TimeZone.getTimeZone("UTC");
Calendar startTime = Calendar.getInstance(UTCTimeZone);
Calendar endTime = Calendar.getInstance(UTCTimeZone);

startTime.add(Calendar.MINUTE, fsMain.CallbackTime.getValue());
endTime.add(Calendar.MINUTE, fsMain.CallbackTime.getValue() +
    fsMain.CallbackDuration.getValue());

wcbRequest.setStartTime(startTime.getTime());
wcbRequest.setEndTime(endTime.getTime());
wcbRequest.setTimeShift(fsMain.TimeShift.getValue());

    Set information regarding email notifications about the new callback
    interaction.

wcbRequest.setDoEmailNotification(fsMain.DoEmailNotifications.getValue());
wcbRequest.setNotificationEmail(fsMain.EmailAddress.getValue());

    Include the customers contact information in the interaction.

KeyValueCollection contactInfo = new KeyValueCollection();
contactInfo.addString("FirstName", fsMain.FirstName.getValue());
contactInfo.addString("LastName", fsMain.LastName.getValue());
contactInfo.addString("EmailAddress", fsMain.EmailAddress.getValue());
if (fsMain.Media.getValue().equals("voice"))

{
    contactInfo.addString("PhoneNumber", fsMain.PhoneNumber.getValue());
}

wcbRequest.setBusinessAttributes(contactInfo);

try
{
    WCBMessage wcbResponse;
    wcbResponse = wcProtocol.Process(wcbRequest, isProtocol);

    if (wcbResponse.getId() == WCBResponseError.ID)
    {
        printErrorMessage(out, ((WCBResponseError) wcbResponse).getErrorMessage());
    }
    else if (wcbResponse.getId() == WCBResponseAck.ID)
    {
        String interactionId = ((WCBResponseAck) wcbResponse).getInteractionId();

        Display a single web callback interaction.

        fsAction.Action.setValue(Actions.ShowSingleCallbackForm);
        fsItxParams.InteractionId.setValue(interactionId); ...

```

Retrieve a Single Web Callback Interaction

The following code retrieves a single web callback interaction using the `WCBRequestGetSingleInteraction` method:

```
if (fsAction.Action.getValue() == Actions.ShowSingleCallbackForm)
{
    WCBMessage wcbResponse;
    WCBRequestGetSingleInteraction wcbRequestGetSingleInteraction =
        WCBRequestGetSingleInteraction.create(fsItxParams.InteractionId.getValue());

    try
    {
        wcbResponse = wcProtocol.Process(wcbRequestGetSingleInteraction, isProtocol);

        if (wcbResponse.getId() == WCBResponseError.ID)
        {
            printErrorMessage(out, ((WCBResponseError) wcbResponse).getErrorMessage());
        }
        else if (wcbResponse.getId() == WCBResponseSingleInteraction.ID)
        {
            WebCallbackInteraction itx = ((WCBResponseSingleInteraction)
                wcbResponse).getWCBInteraction();
            fsItxParams.EmailAddress.setValue(itx.getNotificationEmail());
            fsItxParams.DoEmailNotifications.setValue(itx.getDoEmailNotification());
            fsItxParams.PhoneNumber.setValue(itx.getCustomerNumber());
            fsItxParams.Media.setValue(itx.getDesiredResponseType());
            fsItxParams.TimeShift.setValue(itx.getTimeShift());
            fsItxParams.InteractionId.setValue(itx.getInteractionId());
            fsItxParams.Subject.setValue(itx.getSubject());
        }
    }
}
```

Retrieve a List of Web Callback Interactions

The following code retrieves a list of web callback interactions using the `WCBRequestGetMultipleInteractions` method:

```
if (fsAction.Action.getValue() == Actions.ShowViewListForm)
{
    if (fsMain.PhoneNumber.getValue().equals(""))
    {
        printErrorMessage(out, "Missing 'Phone number (or IP)' value");
    }
    else if (fsMain.PhoneNumber.getValue().contains("") ||
        fsMain.PhoneNumber.getValue().contains("\\"))
    {
        printErrorMessage(out, "Phone number (or IP) contains invalid characters");
    }
    else
    {
        WCBRequestGetMultipleInteractions wcbRequest =
```

```

WCBRequestGetMultipleInteractions.create(fsMain.PhoneNumber.getValue());

try
{
    WCBMessage wcbResponse;
    wcbResponse = wcProtocol.Process(wcbRequest, isProtocol);

    if (wcbResponse.getId() == WCBResponseError.ID)
    {
        printErrorMessage(out, "No interactions found for Phone number (or IP) : " +
            fsMain.PhoneNumber.getValue());
    }
    else if (wcbResponse.getId() == WCBResponseMultipleInteractions.ID)
    {
        WebCallbackInteraction[] interactions = ((WCBResponseMultipleInteractions)
            wcbResponse).getWCBInteractions();

        if (interactions.length == 0)
        {
            printErrorMessage(out, "No interactions found for Phone number (or IP) : "
                + toHtml(fsMain.PhoneNumber.getValue()));...
        }
    }
}

```

Cancel a Web Callback Interaction

The following code cancels a web callback interaction using the `WCBRequestCancel` method and displays the appropriate message:

```

if (fsAction.Action.getValue() == Actions.SubmitCancelCallback)
{
    WCBMessage wcbResponse;
    WCBRequestCancel wcbRequestCancel =
        WCBRequestCancel.create(fsItxParams.InteractionId.getValue());

    try
    {
        wcbResponse = wcProtocol.Process(wcbRequestCancel, isProtocol);

        if (wcbResponse.getId() == WCBResponseError.ID)
        {
            printErrorMessage(out, ((WCBResponseError) wcbResponse).getErrorMessage());
        }
        else if (wcbResponse.getId() == WCBResponseAck.ID)
        {
            %>
            <div class="samplesSampleNameTitle"><p>Web callback was cancelled.</p></div>
            <%

```

Reschedule a Web Callback Interaction

The following code reschedules a web callback interaction using the `WCBRequestReschedule` method:

```
if (fsAction.Action.getValue() == Actions.SubmitRescheduleCallback)
{
    WCBRequestReschedule wcbRequest =
        WCBRequestReschedule.create(fsItxParams.InteractionId.getValue());

    TimeZone UTCTimeZone = TimeZone.getTimeZone("UTC");
    Calendar startTime = Calendar.getInstance(UTCTimeZone);
    Calendar endTime = Calendar.getInstance(UTCTimeZone);

    startTime.add(Calendar.MINUTE, fsItxParams.CallbackTime.getValue());
    endTime.add(Calendar.MINUTE, fsItxParams.CallbackTime.getValue() +
        fsItxParams.CallbackDuration.getValue());

    wcbRequest.setStartTime(startTime.getTime());
    wcbRequest.setEndTime(endTime.getTime());

    wcbRequest.setTimeShift(fsMain.TimeShift.getValue());

    try
    {
        WCBMessage wcbResponse;
        wcbResponse = wcProtocol.Process(wcbRequest, isProtocol);

        if (wcbResponse.getId() == WCBResponseError.ID)
        {
            printErrorMessage(out, ((WCBResponseError) wcbResponse).getErrorMessage());
        }
        else if (wcbResponse.getId() == WCBResponseAck.ID)
        {
            String interactionId = ((WCBResponseAck) wcbResponse).getInteractionId();

            //Show single web callback
            fsAction.Action.setValue(Actions.ShowSingleCallbackForm);
            fsItxParams.InteractionId.setValue(interactionId);
        }...
    }
}
```

Interaction Submit (Genesys 3rd Party Media) Sample

This sample uses a JSP to present a web form from which you can enter information to create, update, and cancel a new interaction. This web form allows you to enter information for Interaction Server, the Interaction ID, and

the media type. It also allows you to enter data that is to be attached to the interaction, such as first name and last name, and three key-value pairs. Fields on the form (host, port ID, and workflow queue) are automatically filled with information that is provided by Configuration Server, but the information can also be filled in manually.

Warning! For this sample to work reliably, the code of your custom application must ensure that each interaction that is generated and submitted by the application has a unique `InteractionID`. Neither the sample code nor Interaction Server error-check for this requirement, so that meeting it is the responsibility your code.

You could choose to allow Interaction Server to generate a unique interaction identifier. In this case, you would design your application to pass an empty `InteractionID`, so that Interaction Server will assign its own `InteractionID` and return that generated ID in the Ack event.

Purpose

The Interaction Submit Sample is a JSP file that shows how to:

- Connect to Interaction Server by using the Interaction API.
- Submit an interaction.
- Update an interaction
- Cancel an interaction.

Functionality Overview

The following sections review the code that is used in the implementation of the different Interaction Submit functions:

- “Retrieving Parameters” on [page 153](#)
- “Getting Instances of Load Balancer and Interaction Server” on [page 154](#)
- “Processing Your Request” on [page 155](#)
- “Closing the Connection” on [page 157](#)
- “Constructing the HTML Body” on [page 157](#)
- “Handling User Events” on [page 159](#)

For information about code that is common to all of the samples—such as retrieving parameters, setting the content type, character encoding, loading libraries, and importing files—see “Common Code” on [page 107](#).

Files

The `.../ItxSubmit` directory contains the Interaction Submit Sample. The sample consists of a single file; `ItxSubmit.jsp`.

Code Explanation

The following subsections explain the code that is in the `ItxSubmit.jsp` file. They appear in the same order as the code in the JSP. In some places, however, several lines of code have been omitted in order to focus your attention on the more important points. In these cases, the missing lines have been replaced with “...”.

Retrieving Parameters

The Java code section now retrieves the request parameters, such as the host name and port name, the first name and the last name, and so on.

Note: The following technique is for demonstration purposes only. Do not use this technique in a production application. If you do not send host and port information directly to the browser, you can reduce the risk of security breaches.

```

svcHost = request.getParameter("host");
if (svcHost == null || svcHost.equals("")) svcHost = "";

String strPort = request.getParameter("port");
if (strPort == null || strPort.equals("")) strPort = "-1";
svcPort = Integer.parseInt(strPort);

String strInteractionID = request.getParameter("interactionid");
if (strInteractionID == null || strInteractionID.equals("")) strInteractionID = "";

String strMediaType = request.getParameter("mediatype");
if (strMediaType == null || strMediaType.equals("")) strMediaType = "email";

String strScriptName = i18nsupport.GetSubmitParametr(request, "scriptname");
if (strScriptName == null || strScriptName.equals("")) strScriptName = "Inbound queue";

String strAction = i18nsupport.GetSubmitParametr(request, "action");
if (strAction == null) strAction = "...";
...
boolean bAgree = false;
String strAgree = i18nsupport.GetSubmitParametr(request, "Agree");

if (strAgree == null || strAgree.equals(""))
    bAgree = false;
else
    bAgree = true;

```

```

int iGender = 0;
String strGender = i18nsupport.GetSubmitParametr(request, "Gender");

if (strGender != null && strGender.equals("Male"))
    iGender = 1;
else if (strGender != null && strGender.equals("Female"))
    iGender = 2;...

```

Getting Instances of Load Balancer and Interaction Server

This section of code creates an instance of Load Balancer and an instance of Interaction Server by using the host and port IDs of Interaction Server. If there are any errors or exceptions, the code handles them:

```

try
{
    ServiceInfo si = null;
    try
    {
        si = LoadBalancer.GetServiceInfo(CfgAppType.CFGInteractionServer, strTenant);
        svcHost = si.getHost();
        svcPort = si.getPort();
    }
    catch (LoadBalancerException e)
    {
        bError = true;
        strError = "We are sorry, Interaction Server is unavailable at this time.
            Please try it again later or enter information manually.";
    }

    strTenantID = LoadBalancer.getTenantId(strTenant);

    kvcAppEndPoints = LoadBalancer.getOwnApp().getOptions().getList
        ("endpoints:" + strTenantID);
    try
    {
        iTenantId = Integer.parseInt(strTenantID);
    }
    catch (Exception e1)
    {
        bError = true;
        strError = "Can't get tenant ID for " + strTenantID + " tenant.
            Please check your configuration.";
    }

    if(svcHost != null && !svcHost.equalsIgnoreCase("") && svcPort != -1 &&
        strAction != null && !strAction.equalsIgnoreCase(""))
    {
        itxProtocol = new InteractionServerProtocol (new Endpoint(new URI("tcp://" +
            svcHost + ":" + String.valueOf(svcPort))));
    }
}

```

```

itxProtocol.setClientType(InteractionClient.MediaServer);
itxProtocol.setClientName("MCR_WebAPIServer8021");

try
{
    itxProtocol.open();
    bConnected = true;
}
catch (Exception e)
{
    strError = "We are sorry, Interaction Server is unavailable at this time.
        Please try again later. " + e.toString();
}...

```

Processing Your Request

This section of code checks to see whether you have selected the Send action. If you have, it collects the parameter data that is sent from the browser in a KeyValueCollection object for transmission to Interaction Server. You will create a RequestSubmit object to send this information to Interaction Server:

```

if (strAction != null && strAction.equals("Send") && bConnected)
{
    try
    {
        RequestSubmit reqSubmit = RequestSubmit.create();
        KeyValueCollection kvcUserData = new KeyValueCollection();
        kvcUserData.addString(attachDataName1, attachDataValue1);
        kvcUserData.addString(attachDataName2, attachDataValue2);
        kvcUserData.addString(attachDataName3, attachDataValue3);
        kvcUserData.addString("FirstName", strFirstName);
        kvcUserData.addString("LastName", strLastName);
        if (bAgree)
            kvcUserData.addInt("Agree", 1);
        else
            kvcUserData.addInt("Agree", 0);
        kvcUserData.addInt("Gender", iGender);

        reqSubmit.setUserData(kvcUserData);
        reqSubmit.setMediaType(strMediaType);
        reqSubmit.setInteractionId(strInteractionID);
        reqSubmit.setQueue(strScriptName);
        reqSubmit.setTenantId(iTenantId);
        reqSubmit.setInteractionType("Inbound");
        reqSubmit.setInteractionSubtype("InboundNew");
        reqSubmit.setIsOnline(Boolean.FALSE);
        Message msg = itxProtocol.request(reqSubmit, 30000);
        if (msg != null && msg instanceof EventAck)
        {
            EventAck eventAck = (EventAck) msg;
            strInteractionID = eventAck.getExtension().getString("InteractionId");
        }
    }
}

```

```

        strError = "Operation successfully submitted to the " + svcHost + ":" + svcPort;
    }
    else if (msg != null && msg instanceof EventError)
    {
        bError = true;
        EventError eventError = (EventError)msg;
        strError = "Can't submit interaction to the InteractionServer. "
            + eventError.getErrorDescription();
    }
    itxProtocol.close();
}...

```

If you have selected the Update action, you must collect the parameter data that is sent from the browser in a `KeyValueCollection` object for transmission to Interaction Server. You will create a `RequestChangeProperties` object to send this information to Interaction Server:

```

else if (strAction != null && strAction.equals("Update") && bError == false)
{
    RequestChangeProperties requestChangeProperties = RequestChangeProperties.create();

    KeyValueCollection kvcDeletedUserData = new KeyValueCollection();
    kvcDeletedUserData.addString(attachDataName3, attachDataValue3);

    KeyValueCollection kvcUpdatedUserData = new KeyValueCollection();
    kvcUpdatedUserData.addString(attachDataName1, attachDataValue1);
    kvcUpdatedUserData.addString(attachDataName2, attachDataValue2);

    kvcUpdatedUserData.addString("FirstName", strFirstName);
    kvcUpdatedUserData.addString("LastName", strLastName);
    if (bAgree)
        kvcUpdatedUserData.addInt("Agree", 1);
    else
        kvcUpdatedUserData.addInt("Agree", 0);
    kvcUpdatedUserData.addInt("Gender", iGender);

    requestChangeProperties.setInteractionId(strInteractionID);
    requestChangeProperties.setAddedProperties(kvcUpdatedUserData);
    requestChangeProperties.setDeletedProperties(kvcDeletedUserData);

    Message msg = itxProtocol.request(requestChangeProperties, 30000);...
}

```

If you have chosen to cancel the interaction, you must create a `ReasonInfo` object to pass in to the `RequestStopProcessing` object to cancel the interaction.

```

else if (strAction != null && strAction.equals("Cancel") && bError == false)
{
    ReasonInfo reason = ReasonInfo.create();
    reason.setReason(123456789);
    reason.setReasonDescription("Put your reason here");
    RequestStopProcessing requestStopProcessing

```

```
RequestStopProcessing.create(strInteractionID, reason);

Message msg = itxProtocol.request(requestStopProcessing, 30000);...
```

Closing the Connection

Finally, the JSP closes the connection to Interaction Server. If your action was successful, you will receive a message to that effect in the Response from Server section of your browser window.

```
...itxProtocol.close();
}

if (strError!= null && strError.equalsIgnoreCase("")) &&
    strAction != null && !strAction.equalsIgnoreCase(""))
{
    strError = "Command '" + strAction + "' successfully processed.
        Please check Interaction Server log for details.";
}
```

Constructing the HTML Body

Next, the JSP includes the HTML code for the creation of the input boxes; a listing of media types from which to choose; the Send, Update, and Cancel buttons; and the field that holds messages from the server:

```
<form id="frm_itx" name="frm_itx" method="post" action="ItxSubmit.jsp">
  <table style="width:100%" cellpadding="0" cellspacing="0">
    <tr>
      <td class="samplesSampleTopToolbar">
        <table >
          <tr >
            <td style="width:617px"
class="samplesSampleNameTitleWhite">Interaction Server sample</td>
            <td style="width:auto" ></td>
          </tr>
        </table>
      </td>
    </tr>
    <tr>
      <td class="samplesSampleParagraph">Submit interaction to Interaction server
with PSDK Java API classes and JSP pages<br/><br/></td>
    </tr>
    <tr>
      <td>
        <table border="0">
          <tr>
            <td class="samplesSampleFieldTitle" >Interaction Server host:</td>
```

```

        <td><input style="width:550px; vertical-align: middle;"
class="samplesFieldStyle" type="text" name="host" value="<%=svcHost%>"></td>
        <td style="width:6px; vertical-align: middle;"><span
class="samplesMandatoryMark">*</span></td>
    </tr>...

...<tr>
    <td class="samplesSampleFieldTitle" ></td>
    <td>
        <select style="width:550px; vertical-align: middle;" name="selMedia"
id="selMedia" size="1" onchange="selMedia_onchange();">
            <option value="email" selected>email</option>
            <option value="chat">chat</option>
            <option value="callback">callback</option>
            <option value="sms">sms</option>
            <option value="fax">fax</option>
            <option value="imchat">imchat</option>
            <option value="video">video</option>
            <option value="voice">voice</option>
            <option value="voip">voip</option>
            <option value="webform">webform</option>
            <option value="webcallback">webcallback</option>
        </select>
    </td>
    <td style="width:6px; vertical-align: middle;"></td>
</tr>
<tr>
    <td class="samplesSampleFieldTitle" >Workflow queue:</td>
    <td>
        <%
            if (kvcAppEndPoints != null && kvcAppEndPoints.size() != 0)
            {
                %>
                <select style="width:550px; vertical-align: middle;" name="scriptname"
id="scriptname" size="1" >
                    <%
                        Enumeration eElements = kvcAppEndPoints.getEnumeration();
                        for (; eElements.hasMoreElements(); )
                        {
                            Object oElement = eElements.nextElement();
                            if (oElement instanceof KeyValuePair)
                            {
                                KeyValuePair kvp = (KeyValuePair)oElement;
                                if (kvp.getValueType() == ValueType.STRING)
                                {
                                    String strOptionName = kvp.getStringKey();
                                    String strOptionValue = kvp.getStringValue();
                                    if (strScriptName.compareToIgnoreCase(strOptionValue) == 0)
                                        out.write("          <option value=\"\" + strOptionValue +
\"\" selected>" + strOptionName + "</option>\r\n");
                                    else

```

```

        out.write("        <option value=\"\" + strOptionValue +
        \"\">" + strOptionName + "</option>\r\n");
    }...

...<tr>
    <td class="samplesSampleFieldTitle" >First name:</td>
    <td><input style="width:550px; vertical-align: middle;"
class="samplesFieldStyle" type="text" name="FirstName"
value="<%=strFirstName%>"></td>
        <td style="width:6px; vertical-align: middle;"></td>
    </tr>...

...<tr >
    <td class="samplesSampleFieldTitle" ></td>
    <td style="width:511px; vertical-align: middle; text-align:right">
        <table>
            <tr align="right">
                <td onclick="javascript:on_send();" class="sampleBlueButton"
onmouseover="javascript:MouseOver(this);"
onmouseout="javascript:MouseOut(this);">Submit</td>
                <td onclick="javascript:on_cancel();"
class="sampleBlueButton" onmouseover="javascript:MouseOver(this);"
onmouseout="javascript:MouseOut(this);">Stop</td>
                <td onclick="javascript:on_update();"
class="sampleBlueButton" onmouseover="javascript:MouseOver(this);"
onmouseout="javascript:MouseOut(this);">Update</td>
                <input type="hidden" id="action" name="action">
            </tr>...

```

Handling User Events

Two JavaScript functions handle user events:

- `window_onload()`—Called every time that the page is loaded. This function currently sets the default media type of the form, but you might also modify it to execute any other tasks that should be carried out whenever the page is loaded.
- `selMedia_onChange()`—Sets the media type to the value that is selected by the user.

Functions

The following functions are also provided:

- `getSelectedOption(opt)`—Takes a collection of options for a field and returns the option that has been selected by the user.
- `selSelection(objControl, value)`—Sets the value of an (`objControl`) document field to `Value`.

Statistical Information Sample

The `...\Statistics` directory contains files for the Statistical Information Sample. This sample demonstrates a web form through which users can select from a list of six predefined statistics and retrieve their current values.

Purpose

The Statistical Information Sample shows how to:

- Connect to Statistics Server by using the Platform SDK Java API.
- Specify and submit a list of statistics.
- Retrieve the current values of selected statistics.

Functionality Overview

The following sections outline the code that is used to implement the Statistical Information Sample:

- “Getting Instances of Load Balancer and Statistics Server” on [page 161](#)
- “Event Handlers” on [page 163](#)
- “Closing Connections” on [page 165](#)
- “Constructing the HTML Body” on [page 165](#)
- “Handling User Events” on [page 166](#)

For information about code that is common to all of the samples—such as retrieving parameters, setting the content type, character encoding, loading libraries, and importing files—see “Common Code” on [page 107](#).

Files

The `...\Statistics` directory contains the Statistical Information Sample. The sample consists of a single file; `StatInfo.jsp`.

Code Explanation

The following subsections explain the code in the `StatInfo.jsp` file.

Getting Instances of Load Balancer and Statistics Server

Warning! The various Web API samples illustrate different ways in which your own applications can communicate with the Load Balancer and store data from it. For example, you can:

- Acquire a new server for each request.
- Acquire services by stored aliases.
- Pass the host and port information of the destination server, unaliased.

The last option, which is shown in this sample, is potentially dangerous. It can reveal aspects of your network infrastructure (such as the name and port of your internal server) to potential attackers. Therefore, Genesys recommends that you *not* use this technique in any front-end application.

```
try
{
    try
    {
        ServiceInfo si = LoadBalancer.GetServiceInfo(CfgAppType.CFGStatServer,
            strTenant);
        ssd.setStrHost(si.getHost());
        ssd.setIPort (si.getPort());
    }
    catch (LoadBalancerException e1)
    {
        ssd.setStrMessages("StatServer is not available at this time. Please try it
            later.");
    }
    ssd.setISelectedStatistic(Integer.parseInt(strSelectedStatistic));
}
catch (Exception e)
{
    ssd.setStrMessages("Error during submit: \r\n" + e.toString());
}

if (strGetStatInfo.compareToIgnoreCase("Get Stat Info")==0)
{
    if(ssd.getStrHost() != null && !ssd.getStrHost().equalsIgnoreCase("") &&
        ssd.getIPort() != -1)
    {
        ssd.setStatServerProtocol(new StatServerProtocol (new Endpoint(new URI("tcp://" +
            ssd.getStrHost() + ":" + String.valueOf(ssd.getIPort()))));
        ssd.getStatServerProtocol().setClientId(777);
        ssd.getStatServerProtocol().setClientName("WebAPISamples802");

        try
        {
```

```

        eventsReceiverThread.start();
        ssd.getStatServerProtocol().open();
        ssd.setBConnected(true);
    }
    catch (Exception e)
    {
        ssd.setBConnected(false);
        ssd.setStrMessages("We are sorry, Stat Server is unavailable at this time. Please
            try again later. " + e.toString());
    }...

```

Next, you must determine which statistic the user has selected from the drop-down list, and call the `RequestStatInfo()` method to submit the selected request to Stat Server and return the resulting value:

```

if (ssd.isBConnected())
{
    switch (ssd.getISelectedStatistic())
    {
        case 1:
            strResult = RequestStatInfo(ssd, strTenant, strChatQueue,
                "WebApiSrvQueue_chat_Total_Distribution_Time", StatisticType.Historical,
                strStatTimeProfile, strStatTimeInterval);
            break;
        case 2:
            strResult = RequestStatInfo(ssd, strTenant, strChatQueue,
                "WebApiSrvQueue_chat_Current_In_Queue", StatisticType.Current,
                strStatTimeProfile, strStatTimeInterval);
            break;
        case 3:
            strResult = RequestStatInfo(ssd, strTenant, strChatQueue,
                "WebApiSrvQueue_chat_Total_Distributed", StatisticType.Historical,
                strStatTimeProfile, strStatTimeInterval);
            break;
        case 4:
            strResult = RequestStatInfo(ssd, strTenant, strChatQueue,
                "WebApiSrvQueue_chat_Current_In_Processing_In_Queue", StatisticType.Current,
                strStatTimeProfile, strStatTimeInterval);
            break;
        case 5:
            strResult = RequestStatInfo(ssd, strTenant, strChatQueue,
                "WebApiSrvQueue_chat_Current_Waiting_Processing_In_Queue",
                StatisticType.Current, strStatTimeProfile, strStatTimeInterval);
            break;
        case 6:
            strResult = RequestStatInfo(ssd, strTenant, strEmailQueue,
                "WebApiSrvQueue_email_Total_Waiting_Time", StatisticType.Historical,
                strStatTimeProfile, strStatTimeInterval);
            break;
        case 7:
            strResult = RequestStatInfo(ssd, strTenant, strEmailQueue,

```

```

        "WebApiSrvQueue_email_Queue_Length", StatisticType.Current,
        strStatTimeProfile, strStatTimeInterval);
    break;
    case 8:
        strResult = RequestStatInfo(ssd, strTenant, strEmailQueue,
        "WebApiSrvQueue_email_Total_Distributed", StatisticType.Historical,
        strStatTimeProfile, strStatTimeInterval);
    break;
    default:
        strResult = "Incorrect selected option.";
    break;
}
ssd.getStatServerProtocol().close();
ssd.setStrMessages(ssd.getStrHost() + ":" + String.valueOf(ssd.getIPort()) +
"\r\nStat result = " + strResult + "\r\n");
}...

```

Event Handlers

The `RequestStatInfo()` method is a public method that is called in the switch statement that is previously documented. It submits the selected statistics request to Stat Server and return the resulting value as a `String`:

```

public String RequestStatInfo(StatServerData ssd, String strTenantName, String
    strQueueName, String strStatMetrics, StatisticType stType, String strStatTimeProfile,
    String strStatTimeInterval)
{
    String strReturnValue = "";
    StatisticObject objectDescription = new StatisticObject(strTenantName, strQueueName,
        StatisticObjectType.StagingArea);
    StatisticMetric statisticMetric = new StatisticMetric(strStatMetrics);
    statisticMetric.setTimeProfile(strStatTimeProfile);
    statisticMetric.setTimeRange(strStatTimeInterval);
    Statistic queueStat = new Statistic(objectDescription, statisticMetric);
    StatisticsCollection statisticsCollection = new StatisticsCollection();
    statisticsCollection.addStatistic(queueStat);

    Notification notification = Notification.create(NotificationMode.Immediate, new
        Integer(1));
    RequestOpenPackage requestOpenPackage = RequestOpenPackage.create(new
        Integer(12345), stType, statisticsCollection, notification);
    Message m = null;
    try
    {
        m = ssd.getStatServerProtocol().request(requestOpenPackage);
    }...
}

```

The `StatServerEventThread` thread is used to handle responses from Stat Server and to notify the main thread about these responses by setting the state of the `lockStatServerReply` object. The main thread waits for these notifications and analyses the response or exits by time-out. You may want to

consider increasing or decreasing the default timeout by 5 seconds depending on the load to your statserver while in production mode.

```

if (m != null)
{
    if (m.messageName().compareToIgnoreCase(EventPackageOpened.NAME) == 0)
    {
        ssd.getLockStatServerReply().acquire(5000);
        if (ssd.getLockStatServerReply().bIsTimeOut)
        {
            strReturnValue = "Timeout occurred. No response from StatServer.";
        }
        else if (ssd.getMsgStatInfoReply() != null
            && ssd.getMsgStatInfoReply().messageName().
                compareToIgnoreCase(EventPackageInfo.NAME) == 0)
        {
            EventPackageInfo epi = (EventPackageInfo) ssd.getMsgStatInfoReply();
            StatisticsCollection sc = epi.getStatistics();
            for (int i = 0; i < sc.getCount(); i++)
            {
                Statistic stat = sc.getStatistic(i);
                strReturnValue += stat.getStringValue() + "\r\n";
            }
        }
        else
        {
            if (ssd.getMsgStatInfoReply() != null)
                strReturnValue = "Error. Unexpected message from StatServer: " +
                    ssd.getMsgStatInfoReply().messageName();
            else
                strReturnValue = "Error. Empty message from StatServer.";
        }
    }
    else if (m.messageName().compareToIgnoreCase(EventPackageError.NAME) == 0)
    {
        EventPackageError epe = (EventPackageError)m;
        strReturnValue = epe.getDescription();
    }
    else
    {
        strReturnValue = "Error. Unexpected message from StatServer: " +
            m.messageName();
    }
}
else
{
    strReturnValue = "Error. Empty message from StatServer.";
}
return strReturnValue; ...

```

Closing Connections

The following line of code closes the connection to Stat Server, otherwise; there will be orphan connections to the server that do not relinquish their resources—principally, network resources (such as sockets) and memory:

```
ssd.getStatServerProtocol().close();
```

Constructing the HTML Body

Next, the JSP includes the HTML code for the creation of the drop-down list that contains statistics for you to display:

```
<form id="frm_stat" name="frm_stat" action="StatInfo.jsp" method="post">
  <table style="width:100%" cellpadding="2">
    <tr>
      <td class="samplesSampleTopToolbar">
        <table>
          <tr>
            <td style="width:617px"
class="samplesSampleNameTitleWhite">Statistics Server Sample</td>
            <td style="width:auto" ></td>
          </tr>
        </table>
      </td>
    </tr>

    <tr>
      <td class="samplesSampleParagraph">Select predefined statistical information
below to proceed.</td>
    </tr>
    <tr>
      <td>
        <table style="width:617px" border="0">
          <tr>
            <td class="samplesSampleFieldTitle" ></td>
            <td></td>
            <td style="width:6px; vertical-align: middle;"></td>
          </tr>
          <tr>
            <td style="width:100px" class="samplesSampleFieldTitle">Statistic
name:</td>
            <td>
              <select name="selStatistic" id="selStatistic">
                <option value="1">Chat: total distribution time</option>
                <option value="2">Chat: queue length</option>
                <option value="3">Chat: total distributed</option>
                <option value="4">Chat: queue-current in processing</option>
                <option value="5">Chat: queue-current waiting processing</option>
```

```

        <option value="6">Webform: total distribution time</option>
        <option value="7">Webform: queue length</option>
        <option value="8">Webform: total distributed</option>
    </select>
</td>
    <td style="width:6px; vertical-align: middle;"><span
class="samplesMandatoryMark">*</span></td>
</tr>

<tr>
    <td style="width:100px" class="samplesSampleFieldTitle" >Response from
server:</td>
    <td><textarea style="height:150px;width:550px; vertical-align: middle;"
class="samplesFieldStyle" name="Messages" id="Messages"
rows="5"><%=mask_html_for_textarea(ssd.getStrMessages())%></textarea></td>
    <td style="width:6px; vertical-align: middle;"></td>
</tr>...

```

Handling User Events

These StatInfo.jsp functions currently have no implementation:

- window_onload()
- selMedia_onChange()

Universal Contact Server Sample

This sample demonstrates a web form through which users can access the interaction history of a given contact from the Interaction table of the UCS database and display it in an ordered list.

Purpose

The Universal Contact Server Sample shows how to:

- Connect to UCS by using the PSKD API.
- Query the database based on first name, last name and e-mail address.
- Display the result of the query.
- Disconnect from the UCS.

Functionality Overview

The following sections outline the code that is used to implement the Universal Contact Server Sample:

- User Interface Implementation:
 - “Declaring Variables” on [page 167](#)

- “Building the Lists of Contacts and Interactions” on [page 168](#)
- “Getting Instances of Load Balancer and UCS” on [page 171](#)
- “Submitting User Information” on [page 172](#)
- “Closing Connections” on [page 173](#)
- “Handling User Events” on [page 173](#)
- “Constructing the HTML Body” on [page 173](#)
- Logic Implementation:
 - “Declaring Variables” on [page 175](#)
 - “Required Methods” on [page 175](#)
 - “Getting Instances of Load Balancer and UCS” on [page 178](#)
 - “Drawing the Form and Submitting User Data” on [page 179](#)
 - “Closing Connections” on [page 180](#)
 - “Handling User Events” on [page 180](#)

For information about code that is common to all of the samples—such as retrieving parameters, setting the content type, character encoding, loading libraries, and importing files—see “Common Code” on [page 107](#).

Files

The `...\UCS` directory contains the Universal Contact Server Sample. The sample consists of two files; `UCS.jsp` and `Action.jsp`.

Code Explanation

The Universal Contact Server Sample separates UI and logic components. The following subsections explain the code that is in the `UCS.jsp` and `Action.jsp` files.

UI Implementation

The `UCS.jsp` file controls most of the page presentation to the user. The following subsections explain the code in that file.

Declaring Variables

The following variables are declared. Most of them represent the color scheme of the sample. You may adjust them according to your own scheme:

```
String threadColorNew           = "#8CB388";
String threadBackgroundColorNew = "#CFECEC";
String threadColorOld           = "#C0C0C0";
String threadBackgroundColorOld = "#DFE8F6";
String baseBoldColor            = "#000080";

String emailFromCustomerColor   = "#DBE1EA";
```

```

String emailFromCustomerBorderColor = "#CFECEC";
String emailFromAgentColor          = "#EFF1F5";
String emailFromAgentBorderColor    = "#EAE8DB";
String bodyColor                     = "#FFFFFF";
String bodyBackgroundColor           = "#000000";

```

Building the Lists of Contacts and Interactions

The `CreateGetContacts(boolean)` method retrieves the list of sorted contacts:

```

private Message CreateGetContacts(boolean whisSort, Integer iTenantID, String
    strFirstName, String strLastName, String strEmail)
{
    RequestGetContacts gcr      = new RequestGetContacts();
    ComplexSearchCriteria csc   = null;
    gcr.setTenantId(iTenantID);
    gcr.setMaxCount(new Integer(5));
    gcr.setRestricted(Boolean.FALSE);
    gcr.setSearchCriteria(new SearchCriteriaCollection());

    if((strFirstName != null) && !strFirstName.equals(""))
    {
        SimpleSearchCriteria simple1 = new SimpleSearchCriteria();
        simple1.setAttrName(ContactSearchCriteriaConstants.FIRST_NAME);
        simple1.setAttrValue(strFirstName);
        simple1.setOperator(Operators.Equal);

        csc = new ComplexSearchCriteria();
        csc.setPrefix(Prefixes.And);
        csc.setCriteria(new SearchCriteriaCollection());
        csc.getCriteria().add(simple1);
        gcr.getSearchCriteria().add(csc);
    }

    if((strLastName != null) && !strLastName.equals(""))
    {
        SimpleSearchCriteria simple2 = new SimpleSearchCriteria();
        simple2.setAttrName(ContactSearchCriteriaConstants.LAST_NAME);
        simple2.setAttrValue(strLastName);
        simple2.setOperator(Operators.Equal);

        csc = new ComplexSearchCriteria();
        csc.setPrefix(Prefixes.And);
        csc.setCriteria(new SearchCriteriaCollection());
        csc.getCriteria().add(simple2);
        gcr.getSearchCriteria().add(csc);
    }...

    ...if (whisSort)
    {

```

```

        gcr.setSortCriteria(new SortCriteriaCollection());
        SortCriteria srt = new SortCriteria();
        srt.setSortIndex(new Integer(0));
        srt.setAttrName(ContactSortCriteriaConstants.FIRST_NAME);
        srt.setSortOperator(SortMode.Ascending);
        gcr.getSortCriteria().add(srt);
    }
    return gcr;
}

```

The `CreateGetInteractions(String)` method retrieves the list of sorted interactions and returns the `RequestGetInteractionsForContact` object that will be sent to the server:

```

private Message CreateGetInteractions(String strContactID)
{
    RequestGetInteractionsForContact rgifc = new RequestGetInteractionsForContact();
    rgifc.setContactId(strContactID);
    rgifc.setAttributeList(new StringList());
    rgifc.getAttributeList().add(InteractionAttributeListConstants.ID);
    rgifc.getAttributeList().add(InteractionAttributeListConstants.TYPE_ID);
    rgifc.getAttributeList().add(InteractionAttributeListConstants.SUBTYPE_ID);
    rgifc.getAttributeList().add(InteractionAttributeListConstants.MEDIA_TYPE_ID);
    rgifc.getAttributeList().add(InteractionAttributeListConstants.SUBJECT);
    rgifc.getAttributeList().add(InteractionAttributeListConstants.TEXT);
    rgifc.getAttributeList().add(InteractionAttributeListConstants.THREAD_ID);
    rgifc.getAttributeList().add(InteractionAttributeListConstants.WEB_SAFE_EMAIL_STATUS);
    rgifc.getAttributeList().add(InteractionAttributeListConstants.START_DATE);

    rgifc.setSearchCriteria(new SearchCriteriaCollection());
    SimpleSearchCriteria searchByMediaType = new SimpleSearchCriteria();
    searchByMediaType.setOperator(Operators.Equal);

    searchByMediaType.setAttrName(InteractionSearchCriteriaConstants.MEDIA_TYPE_ID);
    searchByMediaType.setAttrValue("email");
    rgifc.getSearchCriteria().add(searchByMediaType);

    rgifc.setSortCriteria(new SortCriteriaCollection());

    SortCriteria srt1 = new SortCriteria();
    srt1.setAttrName(InteractionSortCriteriaConstants.THREAD_ID);
    srt1.setSortIndex(new Integer(0));
    srt1.setSortOperator(SortMode.Ascending);
    rgifc.getSortCriteria().add(srt1);

    SortCriteria srt2 = new SortCriteria();
    srt2.setAttrName(InteractionSortCriteriaConstants.START_DATE);
    srt2.setSortIndex(new Integer(1));
    srt2.setSortOperator(SortMode.Descending);
}

```

```

    rgifc.getSortCriteria().add(srt2);
    return rgifc;
}

```

The `BuildInteractionTable(ContactInteractionList)` method creates an HTML table from the list of sorted interactions. Two private methods—`GetAttributesAsHashtable(InteractionAttributeList)` and `GetAttributeAsString(Hashtable, String)`—and two public methods—`GenerateJavaScript(String, String, int)` and `mask_html_with_cr(String)`—are called by the `BuildInteractionTable` method. They can also be found in the `UCS.jsp` file:

```

public String BuildInteractionTable(ContactInteractionList cil)
{
    String strEmailColor          = "";
    String strEmailBorderColor    = "";
    String strLastThreadID        = "";
    String strCurrentThreadID     = "";
    String strSubtypeId           = "";
    String strMediaTypeId         = "";
    String strSubject             = "";
    String strWebSafeEmailStatus  = "";
    int iThreadCounter            = 0;
    String strInteractionState     = "New";
    String strThreadInteractionState = "Read";
    boolean bFromCustomer         = false;
    StringBuilder strTableContent = new StringBuilder("\r\n
<table border=\"0\" width=\"100%\" id=\"threadsTable\" cellpadding=\"5\"
style=\"border-collapse: collapse\">\r\n");
    int iTotalIncluded            = 0;

    for (int i = 0; i < cil.size(); i++)
    {
        ContactInteraction ci = (ContactInteraction) cil.get(i);
        Hashtable htAttributes =
        GetAttributesAsHashtable(ci.getInteractionAttributes());
        strMediaTypeId         = GetAttributeAsString(htAttributes,
        InteractionAttributeListConstants.MEDIA_TYPE_ID);
        strSubtypeId           = GetAttributeAsString(htAttributes,
        InteractionAttributeListConstants.SUBTYPE_ID);
        strWebSafeEmailStatus  = GetAttributeAsString(htAttributes,
        InteractionAttributeListConstants.WEB_SAFE_EMAIL_STATUS);

        if (strSubtypeId.compareToIgnoreCase("OutboundAutoResponse")==0 ||
        strSubtypeId.compareToIgnoreCase("OutboundNew")==0 ||
        strSubtypeId.compareToIgnoreCase("OutboundNotification")==0 ||
        strSubtypeId.compareToIgnoreCase("OutboundReply")==0)
            bFromCustomer = false;
        else
            bFromCustomer = true;
    }
}

```

```

    if (!bFromCustomer)
    {
        strInteractionState = "New";
        if (strWebSafeEmailStatus != "")
            strInteractionState = strWebSafeEmailStatus;
    }
    else
        strInteractionState = "Read";

```

The following conditional statement will allow you to display only e-mail interactions that are retrieved from UCS:

```

if (strMediaTypeId.compareToIgnoreCase("email")==0 &&
    (strSubtypeId.compareToIgnoreCase("InboundNew")==0 ||
    strSubtypeId.compareToIgnoreCase("OutboundAutoResponse")==0 ||
    strSubtypeId.compareToIgnoreCase("InboundCustomerReply")==0 ||
    strSubtypeId.compareToIgnoreCase("OutboundNew")==0 ||
    strSubtypeId.compareToIgnoreCase("OutboundNotification")==0 ||
    strSubtypeId.compareToIgnoreCase("OutboundReply")==0))
{
    iTotalIncluded++;
    strCurrentThreadID = GetAttributeAsString(htAttributes,
InteractionAttributeListConstants.THREAD_ID);
    strSubject = GetAttributeAsString(htAttributes,
InteractionAttributeListConstants.SUBJECT);

    if (strCurrentThreadID.compareToIgnoreCase(strLastThreadID) != 0)
    {
        if (iThreadCounter > 0)
        {
            strTableContent.append("</table></td></tr></table>\r\n");
            if (strThreadInteractionState.compareToIgnoreCase("New")==0)...

```

Getting Instances of Load Balancer and UCS

The next section of code attempts to get instances of the Load Balancer and UCS. If it succeeds, it captures the Interaction Server host and port information, as well as the tenant name, to variables:

```

if (strHost.compareToIgnoreCase("") == 0 || iPort == -1)
{
    try
    {
        ServiceInfo si = LoadBalancer.GetServiceInfo(CfgAppType.CFGContactServer,
            strTenant);
        strHost = si.getHost();
        iPort = si.getPort();
    }
}

```

```

    catch (LoadBalancerException e1)
    {
        strTableContent = "Contact server is not available at this time. Please try it
            later.";
    }
}
else
{
    if (strAction.compareToIgnoreCase("Search")==0)
    {
        ucsp = new UniversalContactServerProtocol(new Endpoint(new URI("tcp://" + strHost +
            ":" + String.valueOf(iPort))));
        ucsp.setClientName("WebAPISamples802");
        boolean bConnected = false;
        try
        {
            ucsp.open();
            bConnected = true;
        }...
    }
}

```

Submitting User Information

When you are connected to UCS, the query information of the user is submitted, and the sorted results are added to the `strTableContent` variable to be displayed:

```

if(bConnected)
{
    Message msg = null;
    Message msgGetContacts = CreateGetContacts(true, iTenantID, strFirstName,
        strLastName, strEMail);

    try
    {
        msg = ucsp.request(msgGetContacts, 30000);
        if (msg != null && msg instanceof EventGetContacts)
        {
            EventGetContacts eventGC = (EventGetContacts)msg;
            ContactDataList cdl = eventGC.getContactData();
            if (cdl != null && (int)eventGC.getCurrentCount().intValue() > 0)
            {
                Contact contact = (Contact)cdl.get(0);
                strContactID = contact.getId();

                msg = ucsp.request(CreateGetInteractions(strContactID), 30000);
                if (msg != null && msg instanceof EventError)
                {
                    EventError error = (EventError)msg;
                    strTableContent = error.getFaultString();
                }
                else if (msg != null && msg instanceof EventGetInteractionsForContact)

```

```

{
    EventGetInteractionsForContact egifc =
        (EventGetInteractionsForContact)msg;
    ContactInteractionList cil = egifc.getContactInteractions();
    if (cil != null)
    {
        Collections.sort(cil, new SpecialComparator());
        strTableContent = BuildInteractionTable(cil);
    }...
}

```

Closing Connections

Finally, this line closes the connection to UCS, so as to avoid resource leakage:

```
ucsp.close();
```

Handling User Events

Nine JavaScript functions handle user events:

- `window_onload()`—Has no code. It is an empty function.
- `RefreshTable()`—Refreshes the e-mail table. It is a callback function from the `Action.jsp` file.
- `on_search()`—Calls `CheckParams()` and submits the form when the search button is clicked.
- `ChangeThreadStatus()`—Changes the background and border colors of a thread.
- `PrintThread()`—Opens a pop-up window that allows you to print an e-mail thread.
- `MarkAsRead()`—Opens a pop-up window that allows you to mark an e-mail message as Read (that is, previously viewed).
- `PrintOneEmail()`—Opens a pop-up window that allows you to print one e-mail message.
- `ClickOnRow()`—Displays a plus (+) or minus (-) symbol next to a selected row in the e-mail list.
- `CheckParams()`—Checks that the required parameters are not empty

Constructing the HTML Body

Next, the HTML code creates the input fields for first name, last name, and e-mail address; creates the Search button; and displays the query results:

```

</script>
<form name="ucs_form" id="ucs_form" method="post" action="UCS.jsp" >
<table width="100%" cellpadding="0" cellspacing="2">
<tr>
<td class="samplesSampleTopToolbar">
<table >

```

```

        <tr >
            <td style="width:617px" class="samplesSampleNameTitleWhite">UCS
Service Sample</td>
            <td style="width:auto" ></td>
        </tr>
    </table>
</td>
</tr>
<tr>
    <td class="samplesSampleParagraph">Please enter your information below to
proceed.</td>
</tr>

<tr>
    <td class="samplesPanelContent" height="100%">
        <table style="width:617px" border="0">
            <tr>
                <td class="samplesSampleFieldTitle" ></td>
                <td></td>
                <td style="width:6px; vertical-align: middle;"></td>
            </tr>...

...<tr>
            <td class="samplesSampleFieldTitle"></td>
            <td align="right">
                <table>
                    <tr>
                        <td runat="server" onclick="javascript:on_search();"
class="sampleBlueButton" onmouseover="javascript:MouseOver(this);"
onmouseout="javascript:MouseOut(this);"
                        <input name="Search" id="Search" type="hidden"
value="Search" />
                        Submit
                    </td>
                </tr>
            </table>
        </td>
        <td style="width:6px; vertical-align: middle;"></td>
    </tr>

    <tr>
        <td colspan="3" class="samplesSampleParagraph">
            Color diagram:
        </td>
    </tr>
    <tr>
        <td colspan="3">
            <table style="width:100%">
                <tr>

```

```

        <td class="samplesSampleFieldTitleFontOnly" style="width:25%;
background-color:<%=threadBackgroundColorNew%>" align="center">New messages</td>
        <td class="samplesSampleFieldTitleFontOnly" style="width:25%;
background-color:<%=threadBackgroundColorOld%>" align="center">No new messages</td>
        <td class="samplesSampleFieldTitleFontOnly" style="width:25%;
background-color:<%=emailFromCustomerColor%>" align="center">Message from
customer</td>
        <td class="samplesSampleFieldTitleFontOnly" style="width:25%;
background-color:<%=emailFromAgentColor%>" align="center">Message from call
center</td>
    </tr>
    <tr>
        <td colspan="4">
            <hr style="width:617px" class="samplesHrSeparator" />
        </td>
    </tr>
</table>
</td>
</tr>...

```

Logic Implementation

The `action.jsp` file contains the code that is needed to allow the user to mark the e-mail messages as read, print the thread, or print the e-mail message. The following subsections explain the code that is in the file:

Declaring Variables

The following variables are declared:

```

String threadColorNew           = "#8CB388";
String threadBackgroundColorNew = "#CCFFFF";
String threadColorOld           = "#C0C0C0";
String threadBackgroundColorOld = "#DFE8F6";
String baseBoldColor            = "#000080";

String emailFromCustomerColor   = "#EFF1F5";
String emailFromCustomerBorderColor = "#DBE1EA";
String emailFromAgentColor      = "#FAFAF5";
String emailFromAgentBorderColor = "#EAE8DB";
String bodyColor                = "#FFFFFF";
String bodyBackgroundColor      = "#000000";

```

Required Methods

The `CreateInteractionUpdateAttributes(string, int)` method updates the status of an e-mail to read:

```
private Message CreateInteractionUpdateAttributes(String strInteractionID,
    int iTenantID)
{
```

In the following code, set the mandatory properties of the
RequestUpdateInteraction() object:

```
RequestUpdateInteraction rui = new RequestUpdateInteraction();
rui.setInteractionAttributes(new InteractionAttributes());
rui.getInteractionAttributes().setId(strInteractionID);
rui.getInteractionAttributes().setTenantId(new Integer(iTenantID));
rui.getInteractionAttributes().setEntityTypeId(EntityTypes.EmailOut);
rui.setEntityAttributes(new EmailOutEntityAttributes());
rui.getInteractionAttributes().setOtherFields(new KeyValueCollection());
```

Now, set the value of the attribute that you want to update:

```
rui.getInteractionAttributes().getOtherFields().
    addObject(InteractionAttributeListConstants.WEB_SAFE_EMAIL_STATUS, "read");
return rui;
}
```

The CreateGetInteractionsForThreadId(String, String) creates an e-mail
history list for a given interaction:

```
private Message CreateGetInteractionsForThreadId(String strContactID,
    String strThreadId)
{
    RequestGetInteractionsForContact rgifc = new RequestGetInteractionsForContact();
    rgifc.setContactId(strContactID);
    rgifc.setAttributeList(new StringList());
```

Set the list of attributes that you want to retrieve from UCS:

```
rgifc.getAttributeList().add(InteractionAttributeListConstants.ID);
rgifc.getAttributeList().add(InteractionAttributeListConstants.TYPE_ID);
rgifc.getAttributeList().add(InteractionAttributeListConstants.SUBTYPE_ID);
rgifc.getAttributeList().add(InteractionAttributeListConstants.MEDIA_TYPE_ID);
rgifc.getAttributeList().add(InteractionAttributeListConstants.SUBJECT);
rgifc.getAttributeList().add(InteractionAttributeListConstants.TEXT);
rgifc.getAttributeList().add(InteractionAttributeListConstants.THREAD_ID);
rgifc.getAttributeList().add(InteractionAttributeListConstants.
    WEB_SAFE_EMAIL_STATUS);
rgifc.getAttributeList().add(InteractionAttributeListConstants.START_DATE);
```

Note: Requesting the InteractionAttributeListConstants.IxnAttributes
attribute might result in a dramatic decrease in the performance of
UCS. Request the attribute only if it is required.

```
rgifc.setSearchCriteria(new SearchCriteriaCollection());
```

Search for only interactions that meet the ThreadID search criteria:

```
SimpleSearchCriteria searchByThreadId = new SimpleSearchCriteria();
searchByThreadId.setOperator(Operators.Equal);
searchByThreadId.setAttrName(InteractionSearchCriteriaConstants.THREAD_ID);
searchByThreadId.setAttrValue(strThreadId);
rgifc.getSearchCriteria().add(searchByThreadId);

rgifc.setSortCriteria(new SortCriteriaCollection());
```

Next, sort and group the interactions first by ThreadID and then by StartDate:

```
SortCriteria srt1 = new SortCriteria();
srt1.setAttrName(InteractionSortCriteriaConstants.THREAD_ID);
srt1.setSortIndex(new Integer(0));
srt1.setSortOperator(SortMode.Ascending);
rgifc.getSortCriteria().add(srt1);

SortCriteria srt2 = new SortCriteria();
srt2.setAttrName(InteractionSortCriteriaConstants.START_DATE);
srt2.setSortIndex(new Integer(1));
srt2.setSortOperator(SortMode.Ascending);
rgifc.getSortCriteria().add(srt2);
return rgifc;
}
```

The `CreateGetInteractionsByEmailId(string, string)` method allows you to print a specific e-mail message:

```
private Message CreateGetInteractionsByEmailId(String strContactID, String strEmailID)
{
    RequestGetInteractionsForContact rgifc = new RequestGetInteractionsForContact();
    rgifc.setContactId(strContactID);
    rgifc.setAttributeList(new StringList());
```

Set the list of attributes that you want to retrieve from UCS:

```
rgifc.getAttributeList().add(InteractionAttributeListConstants.ID);
rgifc.getAttributeList().add(InteractionAttributeListConstants.TYPE_ID);
rgifc.getAttributeList().add(InteractionAttributeListConstants.SUBTYPE_ID);
rgifc.getAttributeList().add(InteractionAttributeListConstants.MEDIA_TYPE_ID);
rgifc.getAttributeList().add(InteractionAttributeListConstants.SUBJECT);
rgifc.getAttributeList().add(InteractionAttributeListConstants.TEXT);
rgifc.getAttributeList().add(InteractionAttributeListConstants.THREAD_ID);
rgifc.getAttributeList().add(InteractionAttributeListConstants.WEB_SAFE_EMAIL_STATUS);
rgifc.getAttributeList().add(InteractionAttributeListConstants.START_DATE);
```

Note: Requesting the `InteractionAttributeListConstants.IxnAttributes` attribute might result in a dramatic decrease in the performance of UCS. Request the attribute only if it is required.

```
rgifc.setSearchCriteria(new SearchCriteriaCollection());
SimpleSearchCriteria searchByThreadId = new SimpleSearchCriteria();
searchByThreadId.setOperator(Operators.Equal);
searchByThreadId.setAttrName(InteractionSearchCriteriaConstants.ID);
searchByThreadId.setAttrValue(strEmailID);
```

Search for the specified e-mail message by the ThreadID:

```
rgifc.getSearchCriteria().add(searchByThreadId);
rgifc.setSortCriteria(new SortCriteriaCollection());
```

To ensure that your results are still sorted, sort the interactions first by ThreadID and then by StartDate:

```
SortCriteria srt1 = new SortCriteria();
srt1.setAttrName(InteractionSortCriteriaConstants.THREAD_ID);
srt1.setSortIndex(new Integer(0));
srt1.setSortOperator(SortMode.Ascending);
rgifc.getSortCriteria().add(srt1);

SortCriteria srt2 = new SortCriteria();
srt2.setAttrName(InteractionSortCriteriaConstants.START_DATE);
srt2.setSortIndex(new Integer(1));
srt2.setSortOperator(SortMode.Descending);
rgifc.getSortCriteria().add(srt2);
return rgifc;
}
```

Getting Instances of Load Balancer and UCS

The next section of code attempts to get instances of the Load Balancer and UCS. If it succeeds, it captures the Interaction Server host and port information, as well as the tenant name, to variables:

```
try
{
    iTenantID = new Integer(Integer.parseInt(LoadBalancer.getTenantId(strTenant)));
}
catch (Exception peTenant)
{
    strTableContent = "Can't parse TenantID number to integer. " + peTenant.getMessage();
}
```

```

try
{
    ServiceInfo si = LoadBalancer.GetServiceInfo(CfgAppType.CFGContactServer,
        strTenant);
    strHost = si.getHost();
    iPort = si.getPort();
}
catch (LoadBalancerException e1)
{
    strTableContent = "Universal Contact server is not available at this time.
        Please try it later.";
}

if (strHost!= null && strHost.compareToIgnoreCase("")!=0 && iPort != -1)
{
    ucsp = new UniversalContactServerProtocol(new Endpoint(new URI("tcp://" + strHost +
        ":" + String.valueOf(iPort))));
    ucsp.setClientName("WebAPISamples802");
    boolean bConnected = false;
    try
    {
        ucsp.open();
        bConnected = true;
    }...
}

```

Drawing the Form and Submitting User Data

The following code is needed to allow the user to mark the e-mail message as read, print the thread, or print the e-mail message. These actions are made possible by using the `CreateInteractionUpdateAttributes`, `CreateGetInteractionsForThreadId`, and `CreateGetInteractionsByEmailId` methods that were previously described. Two private methods—`GetAttributesAsHashtable(InteractionAttributeList)` and `GetAttributeAsString(Hashtable, String)`—and a public method—`mask_html_with_cr(String)`—are also used here and can be found in the `Action.jsp` file:

```

if (bConnected)
{
    Message msg = null;
    if (strAction.compareToIgnoreCase("mark_as_read") == 0)
    {
        msg = ucsp.request(CreateInteractionUpdateAttributes(strEmail_Id,
            iTenantID.intValue()));

        if (msg != null && msg instanceof EventUpdateInteraction)
        {
            strTableContent = "<h5 align=\"center\"
                class=\"samplesSampleNameTitle\">Email has been marked as read.</h5>";...
        }
    }
}

```

When the user refreshes the main window a call is made to a function on another frame. The following code shows how to call another frame:

```
href=\"javascript:window.opener.RefreshTable();close();\">Close this window and refresh
e-mails history.</a>\";
    }
    else if (msg == null)
    {
        strTableContent = "<h5 align=\"center\"
class=\"samplesSampleNameTitle\">Empty response from UCS. Please check UCS log for
details.</h5>\";
    }
    }
    else if (strAction.compareToIgnoreCase("print_thread") == 0)
    {
        msg = ucsp.request(CreateGetInteractionsForThreadId(strContact_Id,
strThread_Id));
        strTableContent = "<h5 align=\"center\"
class=\"samplesSampleNameTitle\">Email thread history</h5>\\r\\n\";
    }
    else if (strAction.compareToIgnoreCase("print_email")==0)
    {
        msg = ucsp.request(CreateGetInteractionsByEmailId(strContact_Id,
strEmail_Id));
        strTableContent = "<h5 align=\"center\"
class=\"samplesSampleNameTitle\">Email info</h5>\\r\\n\";
    }...
}
```

Closing Connections

Finally, this line closes the connection to UCS, so as to avoid resource leakage:

```
ucsp.close();
```

Handling User Events

One JavaScript function handles user events:

- `window_onload()`—Has no code. It is an empty function.

Additional Customizations

This example can be customized further to implement web-safe e-mail functionality by performing the following:

1. Divide the original sample code into two frames.
2. In the first frame:
 - Copy the code from the original JSP page.

- Add the code for the creation of a link or button that will implement “Reply” functionality. For example:

```
<a onClick="javascript:ReplyToEmail78()">Reply</a>
```

3. In the second frame:

- Add the code that is needed for ReplyToEmail78(). In the following code, 78 is the value of the e-mail counter and represents a unique e-mail message in the list:

```
function ReplyToEmail78 ()
{
    parent.WebFormFrame.document.forms[0].Subject.value =
        MaskSymbols("Re: "+strSubject) );
    parent.WebFormFrame.document.forms[0].Body.value =
        MaskSymbolsForReply(strBody);
    parent.WebFormFrame.setReplyAttributes (strParentId);
}
```

Note: To include this e-mail message in the same thread as its parent e-mail message in the UCS database, you must add a ParentId attribute that contains the value of the original e-mail message.

Dynamic Invitation Sample

The ...\\PushChat directory contains files for the Dynamic Invitation Sample. This sample demonstrates how to use a floating pop-up window to provide assistance to your call-center clients. The window offers clients the options to chat with an agent, send an e-mail message, or use Web Self-Service (FAQ).

Purpose

The Dynamic Invitation Sample shows how to:

- Process AJAX requests.
- Check the availability of the required media.
- Display a pop-up window and offer the client the chat, e-mail, or use Web Self-Service (FAQ) options.

Functionality Overview

The following sections outline the code that is used to implement the Dynamic Invitation Sample:

- “Determining Media Availability” on [page 182](#)
- “AJAX Functions” on [page 183](#)
- “Creating and Displaying a Floating Pop-Up Window” on [page 183](#)

- “Drawing the Form and Submitting User Data” on [page 184](#)
- “Handling User Events” on [page 186](#)

For information about code that is common to all of the samples—such as retrieving parameters, setting the content type, character encoding, loading libraries, and importing files—see “Common Code” on [page 107](#).

Files

The `...\PushChat` directory contains the Dynamic Invitation Sample. The sample consists of three files:

- `ajax.js`—Contains the AJAX functionality that is required for sending HTTP requests via `XmlHttpRequest` objects.
- `Available.jsp`—Receives the AJAX requests and checks the availability of the required media before it displays the pop-up screen.
- `Register.jsp`—Contains the code that is required to display the pop-up screen, process the AJAX requests, and control the user activities on a form. Based on the analyses of these activities, an internal JavaScript decides when to display a pop-up window and when to offer chat, e-mail, or Web Self-Serve (FAQ) options to the customer.

Code Explanation

The following subsections explain the code that is in the `Available.jsp`, `Register.jsp` and `ajax.js` files:

Determining Media Availability

The `Available.jsp` file receives AJAX requests and determines if required media are available before it displays the pop-up screen. The following code demonstrates this ability:

Note: This file currently sends a hard-coded response that ensures that the chat media will always be available. Modify the file so that it includes the required environment-communication information and business-processing rules to ensure that all of the required media are currently available.

```
<%
    out.write ("\r\n");
    out.write ("CHAT_AVAILABLE: true,\r\n");
    out.write ("AGENT_AVAILABLE: false\r\n");
    out.write ("")\r\n");
%>
```

AJAX Functions

The `ajax.js` file contains five functions that are required for sending HTTP requests via XMLHttpRequest objects:

- `initReq`—Initialize a request object that is already constructed.
- `getQueryString`—Returns a query String.
- `prepareQueryString`—Creates a query String that is to be passed to the `httpRequest` function.
- `httpRequest`—Creates an HTTP request that can be sent via an XMLHttpRequest object; called in the `initReq` function.
- `createXMLHttpRequest`—Creates an XMLHttpRequest request.

Creating and Displaying a Floating Pop-Up Window

The `Register.jsp` file contains the code that creates and displays the floating pop-up window:

```
<div id="floatdiv" style="z-index:10;display:none;border: 1px solid rgb(34, 102, 170);
padding: 1px; background: rgb(255, 255, 255) none repeat scroll 0% 0%; position:
absolute; width: 250px; height: 150px; -moz-background-clip: -moz-initial;
-moz-background-origin: -moz-initial; -moz-background-inline-policy: -moz-initial;
top: 1941px; left: 1900px;">
```

The following code displays a picture of an agent and prompts the user to either chat with an agent, use Web Self-Serve, or close the pop-up window:

```
<table cellspacing=0 cellpadding=0 width=100% border=0>
  <tr class="samplesPanelHeader">
    <td colspan="2" align="center" >What would you like?</td>
  </tr>

  <tr bgcolor="#FFFFFF">
    <td>
      
    </td>
    <td align="left">
      <ul class="samplesUL">
        <li class="samplesMenuItem">
          <table border="0">
            <tr>
              <td valign="top">
                :: <a href="javascript:StartChatFrame();" title="Chat"
class="samplesFieldLabels" >Chat with an agent</a>
              </td>
            </tr>
          </table>
        </li>
      </ul>
    </td>
  </tr>
</table>
```

```

<li class="samplesMenuItem">
<table border="0">
<tr>
<td valign="top">
:: <a href="javascript:StartWebCallbackFrame();" title="Callback"
class="samplesFieldLabels" >Request Web callback</a>
</td>
</tr>
</table>
</li>

<li class="samplesMenuItem">
<table border="0">
<tr>
<td valign="top">
:: <a href="javascript:StartFAQFrame();" title="FAQ"
class="samplesFieldLabels" >FAQ system</a>
</td>...

```

Note: The JSFX_FloatDiv function that is used to make the pop-up window float is not documented here, but it is included in the Register.jsp file. This function has been obtained from the <http://www.javascript-fx.com> website and must be documented as such if you plan to use it in your code.

Drawing the Form and Submitting User Data

The Register.jsp file contains the code that creates the data-input fields, drop-down boxes, and buttons on the form:

```

<form action="Register.jsp" name="regform" onsubmit="return form_test(this);"
method="post">
<table style="width:100%" cellpadding="2">
<tr>
<td class="samplesSampleTopToolbar">
<table>
<tr>
<td style="width:617px"
class="samplesSampleNameTitleWhite">Dynamic Invitation Sample</td>
<td style="width:auto"></td>
</tr>
</table>
</td>
</tr>

<tr>
<td>
<table border="0">

```

```

<tr>
  <td colspan="2"></td>
</tr>
<tr>
  <td colspan="2" class="samplesSampleParagraph">
    Please select your registration type below.
    If you have a promotional code, please enter that as well.
  </td>
</tr>

```

The Registration Type drop-down box is created in the following code:

```

<tr>
  <td class="samplesSampleFieldTitle" >Registration Type:</td>
  <td>
    <select class="samplesFieldStyle" size="1" name="attendee_type"
id="attendee_type" onfocus="element_onfocus(this);">
      <option value="" selected="selected">Please select...</option>
      <option value="NA">Attendee</option>
      <option value="SE">Sponsor / Exhibitor</option>
      <option value="GE">Genesys Employee / Genesys Speaker</option>
      <option value="SP">Speaker (Non-Genesys Employee)</option>
      <option value="PF">Partner Forum Only</option>
      <option value="PA">Press / Analyst</option>
      <option value="BA">Board of Advisors</option>
    </select>
  </td>
</tr>

```

Next, user data is gathered—including a Promotional Code, if one exists:

```

<tr>
  <td class="samplesSampleFieldTitle" >First name:</td>
  <td><input style="width:550px; vertical-align: middle;"
class="samplesFieldStyle" type="text" name="FirstName" value=""
onfocus="element_onfocus(this);" /></td>
</tr>...

<tr>
  <td class="samplesSampleFieldTitle" >Promotional Code:</td>
  <td><input style="width:550px; vertical-align: middle;"
class="samplesFieldStyle" type="text" name="PromotionalCode" value=""
onfocus="element_onfocus(this);" /></td>
</tr>...

```

Finally, the Submit button is created:

```

<tr>
    <td class="samplesSampleFieldTitle"></td>
    <td align="right">
        <table>
            <tr>
                <td runat="server"
onclick="javascript:document.forms[0].submit();" class="sampleBlueButton"
onmouseover="javascript:MouseOver(this);"
onmouseout="javascript:MouseOut(this);">Submit</td>
            </tr>
        </table>
    </td>
</tr>...

```

Handling User Events

The `Register.jsp` file contains 12 functions that are used for handling user events:

- `window_onkeydown`—Controls the user activities that relate to the keyboard. If the user presses the Backspace or Delete key 10 or more times, the pop-up window is displayed immediately. You can modify this function to monitor any keys that you choose. You should choose keys that might indicate that the user is frustrated and in need of online support. If the keys are not pressed the required number of times, the function resets the value of the default pop-up interval to 15 seconds.
- `element_onfocus`—Controls when a form element gets the focus. If any form element gets the focus more than three times, the pop-up window is displayed immediately. Usually, frustrated users move from element to element more frequently, and change their focus more often than users who are not frustrated.
- `ShowChatOption`—Sends AJAX requests to the server to the `Available.jsp` file in which media availability is determined. The response is processed in the `handleResponse()` callback function, which is described in the next paragraph.
- `handleResponse`—Handles the AJAX response from the server.
- `processTranscript`—Processes the JavaScript that is included in the `Available.jsp` file. It checks the value of the `CHAT_AVAILABLE` variable. If this function is set to `true`, the pop-up windows is displayed. Currently, this function is coded so that the pop-up window is always displayed, even if the media are not available. Modify this function to meet your requirements.
- `HideMenu`—Hides the pop-up menu.
- `StartChatFrame`—Starts the chat sample.
- `StartFAQFrame`—Starts the FAQ sample.
- `window_onload`—Initializes the form.

- `window_onunload`—Has no code. It is an empty function.
- `form_test`—Is not currently implemented.
- `JSFX_FloatDiv`—Floats the pop-up window. The source code for the function was obtained at from www.javascript-fx.com website.

Note: You may use the `JSFX_FloatDiv` function, provided that the following credit remains in the code:

- Floating Div from <http://www.javascript-fx.com>.
-

FAQ (Web Self-Serve) Sample

The section explains the code of the FAQ (Web Self-Serve) Sample.

Purpose

The FAQ (Web Self-Serve) Sample code demonstrates Knowledge Serve and a basic FAQ Service.

Functionality Overview

The following sections outline the code that is used to implement the FAQ (Web Self-Serve) Sample:

- “Handling User Events” on [page 188](#)
- “Building a Category List” on [page 188](#)
- “Processing and Updating Accuracy Feedback” on [page 188](#)
- “Constructing the HTML Body” on [page 189](#)
- “Creating Accuracy Radio Buttons” on [page 190](#)
- “Passing Information Between Forms” on [page 191](#)
- “Obtaining a List of FAQs” on [page 192](#)

For information about code that is common to all of the samples—such as retrieving parameters, setting the content type, character encoding, loading libraries, and importing files—see “Common Code” on [page 107](#).

Files

The `.../FAQ` directory contains the FAQ (Web Self-Serve) Sample. The sample consists of a single file; `FAQ.jsp`.

Code Explanation

Handling User Events

The `faq.jsp` file contains six functions that used to handle user events:

- `buildCategoryList()`—Builds a list of all categories.
- `getCategoryList()`—Gets a category list for the provided category and list.
- `getCategoryDescription()`—Gets a category description for the provided category.
- `getResponse()`—Gets a response by using the list of all categories.
- `getCategory()`—Gets a category by using the list of all categories.
- `getByName()`—Gets a child from the Root Category by name.

Building a Category List

The `buildCategoryList()` method is used to create a list of categories from which to choose:

```
try
{
    wss = Genesys.webapi.media.faq.direct._faq_init.get_faq_root();
    _faq_root wss = Genesys.webapi.media.faq.direct._faq_init.get_faq_root();
    buildCategoryList();
}
```

Processing and Updating Accuracy Feedback

The following code processes the accuracy feedback. The FAQ API updates the accuracy rating of the answer to the original question, based on the selected accuracy radio button:

```
...if (accuracy != null && accuracy.equals("") == false)
{
    _faq_category ctg = getCategory(id);
    double ac = Double.parseDouble(accuracy);
    if (ctg != null)
        wss.add_feedback(question, ctg, ac);

    //Lets show the initial screen after rating
    id = "";
    typeOfRequest = "";
}

if ((id == null || id.equals("")) && (typeOfRequest == null ||
    typeOfRequest.equals("")))
{...
```

Constructing the HTML Body

The following code creates a form that has a drop-down list of categories from which to choose, as well as a button to retrieve the FAQs that relates to the selected category. The form also provides an input box for your direct questions, as well as a button for submitting your questions. This button will retrieve all of the FAQs in the selected category that relate to your question. In addition to the list of FAQs, it will provide a number that represents the confidence score (ranging from 0 to 100) that the FAQ relates to your direct question:

```
<form method="post" action="FAQ.jsp" >

  <table width="100%" cellpadding="2" cellspacing="0">
    <tr>
      <td class="samplesSampleTopToolbar">
        <table>
          <tr>
            <td style="width:617px" class="samplesSampleNameTitleWhite">Web
Self Service example.</td>
            <td style="width:auto" ></td>
          </tr>
        </table>
      </td>
    </tr>

    <tr>
      <td class="samplesSampleParagraph">Web Self Service example. Demonstrates
Knowledge Server FAQ API functionality.</td>
    </tr>

    <tr>
      <td class="samplesPanelContent">

        <table cellpadding="0" cellspacing="0">

          <tr>
            <td class="samplesSampleFieldTitle" >Select a scope:</td>
            <td>
              <select name="ctg" id="ctg">
                <option selected="selected">All Categories</option>
                <%
                  List ct = wss.get_root_category().get_subcategory_list();
                  for (Iterator i = ct.iterator(); i.hasNext(); )
                  {
                    _faq_category item = (_faq_category)i.next();
                    out.println("<option >" + item.get_name() + "</option>");
                  }
                %>
              </select>
            </td>
          </tr>
        </table>
      </td>
    </tr>
  </table>
</form>
```

```

        </td>
        <td style="width:6px; vertical-align: middle;"><span
class="samplesMandatoryMark">*</span></td>
    </tr>

    <tr>
        <td class="samplesSampleFieldTitle"></td>
        <td align="left">
            <table>
                <tr>
                    <td runat="server"
onclick="javascript:document.forms[0].submit_type.value='Get
FAQ';document.forms[0].submit();" class="sampleBlueButton"
onmouseover="javascript:MouseOver(this);" onmouseout="javascript:MouseOut(this);">
                        <input type="hidden" name="submit_type"
id="submit_type" value="">
                        Get FAQ
                    </td>
                </tr>...
            </table>
        </td>
    </tr>...

```

Creating Accuracy Radio Buttons

When you have made a selection from this list, you are presented with the answer, as well as a group of radio buttons that you can use to provide feedback about the accuracy of the answer:

```

if (question != null && question.equals("") == false)
{
    %>
    <tr>
        <td class="samplesSampleTopToolbar" colspan="3">
            <table >
                <tr >
                    <td style="width:617px" class="samplesSampleNameTitleWhite">
                        Please rate the answer.
                    </td>
                    <td style="width:auto" ></td>
                </tr>
            </table>
        </td>
    </tr>

    <tr><td class="samplesSampleFieldTitle" >Original question:</td>
        <td><textarea class="samplesFieldStyle" readonly name="question" rows="3"
cols="5" style="height:75px; vertical-align:
middle;"><%=question%></textarea></td></tr>
    <tr><td colspan="3">&nbsp;</td></tr>

    <tr>
        <td class="samplesSampleFieldTitle" >Accuracy level:</td>

```

```

        <td>
            <form method="post" action="FAQ.jsp">
                <input type="hidden" value="<%=id%>" name="id">
                <table id="RadioBtnList_Resolution"
class="samplesSampleFieldTitleFontOnlyFontOnly" border="0"
style="width:200px;vertical-align: middle;">
                    <tr>
                        <td>
                            <input type="radio" value="-1" name="accuracy"
id="accuracy_0" />
                            <label for="accuracy_0"
class="samplesSampleFieldTitleFontOnly">-10</label>
                        </td>
                        <td>
                            <input type="radio" value="-1" name="accuracy"
id="accuracy_1" />
                            <label for="accuracy_1"
class="samplesSampleFieldTitleFontOnly">-5</label>
                        </td>
                        <td>
                            <input type="radio" value="-1" name="accuracy"
id="accuracy_2" />
                            <label for="accuracy_2"
class="samplesSampleFieldTitleFontOnly">0</label>
                        </td>
                        <td>
                            <input type="radio" value="-1" name="accuracy"
id="accuracy_3" />
                            <label for="accuracy_3"
class="samplesSampleFieldTitleFontOnly">5</label>
                        </td>
                        <td>
                            <input type="radio" value="-1" name="accuracy"
id="accuracy_4" />
                            <label for="accuracy_4"
class="samplesSampleFieldTitleFontOnly">10</label>
                        </td>
                    </tr>
                </table>
            </form>...

```

Passing Information Between Forms

It is important to note that because the original question is entered on the first page of the web form, and the answer is rated on the third page, you have to pass the original question in either a hidden form field or a read-only text area. This complicates the example slightly. If you do not want to implement the feedback feature, remove the code that relates to this functionality. The

following code shows an example of how the original question is passed from one page to the next in a read-only text area:

```
<tr>

    <td class="samplesSampleFieldTitle" >Original question:</td>

    <td>
        <form method="post" action="FAQ.jsp">
            <textarea class="samplesFieldStyle" readonly="readonly"
name="question" rows="3" cols="10" style="height:75px;vertical-align:
middle;"><%=question%></textarea>
            <input type="hidden" value="" name="id" />
        </form>
    </td>
</tr>
```

Obtaining a List of FAQs

In the following code, the `get_faq_list()` method is used to obtain a list of the FAQs:

```
// Request for FAQ List
    %>
    <table width="100%" cellpadding="2">
        <tr>
            <td class="samplesSampleTopToolbar">
                <table >
                    <tr >
                        <td style="width:617px"
class="samplesSampleNameTitleWhite">
                            Web Self Service example.
                            <% if (ctgName != null)
                                out.write(ctgName);
                            %>
                        </td>
                        <td style="width:auto" ></td>
                    </tr>
                </table>
            </td>
        </tr>

        <tr>
            <td>
                <table border="0">
                    <tr>
                        <td>
                            <%
//out.println("<h1>FAQ List</h1>");
List faq = wss.get_faq_list(current_ctg, 0);
```

```

        out.println("        <ul class=\"samplesUL\">");
        for (Iterator i = faq.iterator(); i.hasNext(); )
        {
            _faq_result item = (_faq_result) i.next();
        %>
        <li class="samplesMenuItem">
        <table border="0">
        <tr>
        <td valign="top">

        <%
            out.println("<a href=\"FAQ.jsp?id=" + ctgList.indexOf(item.get_category()) +
\"\">" + getCategoryDescription(item.get_category()) + " (" + item.get_frequency() +
") </a>");
        %>
        </td>
        </tr>
        </table>
        </li>...

```

Media Availability Sample

The ...\\MediaAvailability directory contains the files for the Media Availability Sample. This sample demonstrates how to check which media types are currently available to you.

Purpose

The Media Availability Sample shows how to determine the availability of each media type and display it in a table.

Functionality Overview

The following sections outline the code that is used to implement the Media Availability Sample:

- “Generating a Table Row” on [page 194](#)
- “Checking Media-Type Availability” on [page 195](#)
- “Creating Media-Type Table” on [page 196](#)

For information about code that is common to all of the samples—such as retrieving parameters, setting the content type, character encoding, loading libraries, and importing files—see “Common Code” on [page 107](#).

Files

The ...\\MediaAvailability directory contains the Media Availability Sample. The sample consists of a single file; MediaAvailability.jsp.

Code Explanation

The following subsections explain the code that is in the MediaAvailability.jsp file.

Generating a Table Row

The GenerateTableRow method checks the availability of the media type by calling the IsMediaAvailable method and then creates a row to add to the media-type table. This row displays the name and description of the media type. It also indicates if the media type is available.

```
public void GenerateTableRow(CfgAppType capType, String strTenant, String strURL,
    String strCaption, String strDescription)
{
    String strOut = "";
    boolean bIsMediaAvailable = false;
    if (CfgAppType.CFGNoApplication == capType)
        bIsMediaAvailable = true;
    else
        bIsMediaAvailable = IsMediaAvailable(capType, strTenant);
    if (bIsMediaAvailable)
    {
        strOut += " <tr>\r\n";
        strOut += " <td style=\"width:20px\"><img
src=\"../Resources/Images/media_connect_small.png\" alt=\"\" /></td>\r\n";
        strOut += " <td style=\"width:100%\"><a href=\"\" + strURL + "\"
class=\"samplesMediaAvailabilityHREF\">\" + strCaption + "</a></td>\r\n";
        strOut += " </tr>\r\n";
        strOut += " <tr>\r\n";
        strOut += " <td style=\"width:20px\"></td>\r\n";
        strOut += " <td class=\"samplesMediaAvailabilityDescription\">\" +
strDescription + "</td>\r\n";
        strOut += " </tr>\r\n";
        strConnectedMediaTable += strOut;
        iConnectedMediaCounter++;
    }
    else
    {
        strOut += " <tr>\r\n";
        strOut += " <td style=\"width:20px\"><img
src=\"../Resources/Images/media_disconnect_small.png\" alt=\"\" /></td>\r\n";
        strOut += " <td style=\"width:100%\"><a href=\"\" + strURL + "\"
class=\"samplesMediaAvailabilityHREF\">\" + strCaption + "</a></td>\r\n";
```

```

        strOut += "    </tr>\r\n";
        strOut += "    <tr>\r\n";
        strOut += "        <td style=\"width:20px\"></td>\r\n";
        strOut += "        <td class=\"samplesMediaAvailabilityDescription\">" +
strDescription + "</td>\r\n";
        strOut += "    </tr>\r\n";
        strDisconnectedMediaTable += strOut;
        iDisconnectedMediaCounter++;
    }
    return;
}

```

Checking Media-Type Availability

The `IsMediaAvailable` method uses the `LoadBalancer.GetServiceInfo` method to determine if the media type is available:

```

public boolean IsMediaAvailable (CfgAppType capType, String strTenant)
{
    boolean bReturn = false;
    ServiceInfo si = null;
    try
    {
        si = LoadBalancer.GetServiceInfo(capType, strTenant);
    }
    catch (LoadBalancerException lbe)
    {
    }

    if (si == null || si.getHost() == null || si.getHost().equalsIgnoreCase("")
        || si.getPort() == -1)
    {
        return bReturn;
    }
    else
    {

```

Chat Server is real-time media. The following code checks the current day and time, assuming that chat is available during specific working hours (for example, Monday thru Friday, 10 AM to 6 PM). Modify the code for your specific business needs or eliminate it completely if your Agents are available 7 days a week, 24 hours a day:

```

    if (capType == CfgAppType.CFGChatServer)
    {
        Calendar calCurTime = Calendar.getInstance();
        calCurTime.setTime (new Date(System.currentTimeMillis()));
        int iCurHour = calCurTime.get (Calendar.HOUR_OF_DAY);
        int iCurDayOfWeek = calCurTime.get (Calendar.DAY_OF_WEEK);

```

```

        if (iCurDayOfWeek != Calendar.SUNDAY & iCurDayOfWeek != Calendar.SATURDAY)
            if (iCurHour >= 10 && iCurHour <= 18)
                bReturn = true;
        }
        else
            bReturn = true;
    }

    return bReturn;
}

```

Creating Media-Type Table

The generated table rows are then added to the table:

```

...<body onload="javascript:window_onload();" class="samplesBody">
<table style="width:100%" cellpadding="0" cellspacing="0" border="0">
    <tr>
        <td colspan="2" class="samplesSampleTopToolbar">
            <table>
                <tr>
                    <td colspan="2" class="samplesSampleNameTitleWhite">Media
Availability</td>
                </tr>
            </table>
        </td>
    </tr>
</table>

<table style="width:789px" cellspacing="0" cellpadding="0">
    <tr style="width:789px">
        <td colspan="2" class="samplesSampleParagraph">Media availability demonstrates
how to check which media is available at the moment.</td>
    </tr>...

```

```

GenerateTableRow (CfgAppType.CFGEmailServer,    strTenant, "../Email/Email.jsp",
    "E-mail over the Web", "Sample of E-mail submission over the web based on the PSDK
Java API classes and JSP pages.");
GenerateTableRow (CfgAppType.CFGEmailServer,    strTenant,
    "../EmailWithAttachment/Email.jsp",    "E-mail over the Web sample with file
attachments", "Sample of E-mail submission over the web based on the PSDK Java API
classes and JSP pages. Allows to attach files with form.");
GenerateTableRow (CfgAppType.CFGChatServer,    strTenant,
    "../Chat/HtmlChatFrameSet.jsp",    "Chat sample with "user typing"
notification", "Chat sample based on PSDK Java API classes and JSP pages.");
GenerateTableRow (CfgAppType.CFGChatServer,    strTenant,
    "../AdvancedChat/HtmlChatFrameSet.jsp", "Chat sample with emoticons and hyper-links
support", "Chat sample based on PSDK Java API classes and JSP pages.");
GenerateTableRow (CfgAppType.CFGChatServer,    strTenant,
    "../ChatWidget/ChatWidget.html",    "Chat Widget", "Chat Widget for simple chat
customization.");...

```

Cobrowse Samples Overview

This section outlines files common to all the Cobrowse Samples.

Common Files

The Basic Cobrowse, the Cobrowse with Initial Startup Page, the Cobrowse with Meet Me, and the Chat and Cobrowse samples all use certain common files:

- `hbmessaging.js`—A JavaScript file that contains the API for Cobrowsing Server.
- `qstring.js`—A JavaScript file that contains a utility class.
- `hbmessage_to_var.js`—A messaging file that provides messages from the cobrowse frame to the web-application frame.
- `blank.html`—The default blank page for initializing empty frames that are used by the API.
- `hbapi.html`—Supports the client-side API and the cobrowse applet.
- `hbmessagingform.html`—Increases security by sending agent login information as an HTTP POST request, instead of as a GET request.
- `hbmessage_to_var.html`—A messaging file that provides messages from the cobrowse frame to the web-application frame.

Warning! The `qstring.js`, `hbmessage_to_var.js`, and `hbmessage_to_var.html` files must all reside in the same directory.

For background information about the purposes of these common files, refer to the KANA Response Live documentation that is listed in “Related Documentation Resources” on [page 363](#). Genesys licenses certain Response Live (formerly Hipbone) cobrowsing components from KANA Software, Inc.

Basic Cobrowse Sample

The `...\CoBrowse` directory contains the files for the Basic Cobrowse Sample. This sample demonstrates Basic Cobrowse functionality.

Purpose

The Cobrowse Sample code demonstrates basic cobrowse functionality that uses the Cobrowsing Server API.

Functionality Overview

The following sections outline the code that is used to implement the Basic Cobrowse Sample:

- “Getting Instances of Load Balancer and Cobrowse Server” on [page 198](#)
- “Drawing the Form” on [page 199](#)
- “Declarations” on [page 200](#)
- “Event Handlers” on [page 201](#)

For information about code that is common to all of the samples—such as retrieving parameters, setting the content type, character encoding, loading libraries, and importing files—see “Common Code” on [page 107](#).

Files

The `...\CoBrowse` directory contains the Basic Cobrowse Sample. This sample includes two files beyond those that are covered in the preceding “Common Files” on [page 197](#) section:

- `CoBrowse.htm`—Sets up the display frame.
- `CoBrowseEventHandler.jsp`—Contains the logic of the sample.

Code Explanation

The following subsections explain the code that appears in the `CoBrowseEventHandler.jsp` file.

Getting Instances of Load Balancer and Cobrowse Server

The initial try-catch script block of the `CoBrowseEventHandler.jsp` file creates an instance of Load Balancer and attempts to discover a Cobrowse Server host:

```
try
{
    ServiceInfo si = LoadBalancer.GetServiceInfo(CfgAppType.CFGCoBrowsingServer,
        strTenant);
    if (si != null)
        CoBrowseServerHost = si.getHost();
}
catch(Exception ex)
{
    //Error load balancing is disabled.
    CoBrowseServerHost = null;
    ...
}
```

If Load Balancer does not provide a Cobrowse Server, you obtain a server from the `constants.jsp` file:

```

if(CoBrowseServerHost != null)
    //From Loadbalancer
    out.println("CobrowseHostName = \"\" + CoBrowseServerHost + "\"");
else
    //From constants.jsp
    out.println("CobrowseHostName = \"\" + strConavServerUrl + "\"");
    //From constants.jsp
    out.println("var ConavigationiChannelID = \"\" + strConavChannelID + "\"");...

```

Drawing the Form

This HTML code block draws a page, defines a JavaScript function that reports the host of Cobrowse Server, and defines links to perform basic cobrowse functions:

```

<form name="InfoForm" id="InfoForm" onsubmit="return false;">
    <table style="width:100%" cellpadding="2">
        <tr>
            <td class="samplesSampleTopToolbar">
                <table>
                    <tr>
                        <td style="width:435px"
class="samplesSampleNameTitleWhite">Basic Cobrowse Sample</td>
                        <td style="width:91px"
onclick="javascript:CoBrowse_onclick();" class="sampleGreenButtonBig"
onmouseover="javascript:MouseOver(this);"
onmouseout="javascript:MouseOut(this);">Start</td>
                        <td style="width:91px"
onclick="javascript:EndCoBrowse_onclick();" class="sampleRedButtonBig"
onmouseover="javascript:MouseOver(this);"
onmouseout="javascript:MouseOut(this);">Stop</td>
                        <td style="width:auto" ></td>
                    </tr>
                </table>
            </td>
            <td>
                <tr>
                    <td class="samplesSampleParagraph">Basic cobrowse example. Demonstrates
how to cobrowse with an agent.</td>
                </tr>
                <tr>
                    <td height="100%">
                        <table style="width:617px" border="0">
                            <tr>
                                <td class="samplesSampleFieldTitle" ></td>
                                <td></td>
                                <td style="width:6px; vertical-align: middle;"></td>
                            </tr>...

```

Declarations

This block of JavaScript then declares and initializes core variables and functions. The `window_onunload()` function is declared, but it is not implemented:

```
<script language="javascript">
var HBApiWindow = null;
var HBUserID = "";
var IsFirstConavigation = true;
var IsAgentJoined = false;

var UserLoggedIn = false;
var ConnectTo = "";
var StartPage = "http://www.google.com";

function window_onload()
{
    AddMessage ("Cobrowse server host name: " + CobrowseHostName);
}

function window_onunload ()
{
}
```

The `AddMessage()` function is declared here, and it is used throughout “Event Handlers” on [page 201](#) to display informational and diagnostic messages.

```
function AddMessage (str)
{
    document.forms[0].Messages.value = document.forms[0].Messages.value + "\r\n" + str;
    document.forms[0].Messages.scrollTop = document.forms[0].Messages.scrollHeight;
}
```

The next block of code checks for redundant logins, and then it calls two functions: `HBInitializeAPI()` for initializing the Cobrowse API, and `HBLoginGuest()` for logging in the user properly. These functions are defined in the `hbmessaging.js` file. For details, refer to the KANA Response Live documentation that is listed in “Related Documentation Resources” on [page 363](#).

```
//----- All about Cobrowse -----
function CoBrowse_onclick()
{
    if(UserLoggedIn == true)
    {
        alert("Please log out user "+HBUserID+" before starting a new cobrowse session.");
    }
}
```

```

else
{

if (HBApiWindow == null)
{
    HBApiWindow = parent.hbapi;
    var strUrl = new String(window.location.href);
    var strProcessorUrl =
        strUrl.substring(0, strUrl.lastIndexOf("/"))+"/hbmessage_to_var.html";

    HBApiWindow.HBInitializeAPI("EventHandlerFrame", CobrowseHostName, strProcessorUrl);
    HBApiWindow.HBLoginGuest(ConavigationiChannelID, "", ConnectTo, false,
        "acctSpecificData");
}
else
{
    HBApiWindow.HBLoginGuest(ConavigationiChannelID, "", ConnectTo, false);
}

AddMessage("Connecting to Cobrowse server...");...

```

Note the calls to `AddMessage()` above, as well as in the following calls to the external `HLogout()` function. `HLogout()` is another function that is defined in the `hbessaging.js` file and described in the KANA Response Live documentation.

```

function EndCoBrowse_onclick()
{
    if (HBApiWindow != null)
        HBApiWindow.HLogout();
}

function doExitSession()
{
    AddMessage("Logging out...");
    HBApiWindow.HLogout();
}

```

Event Handlers

This final section of code defines event handlers for cobrowse requests. Each of these event-handler functions typically calls a function of the same name in the `hbessaging.js` file. For details about those external functions, refer to the KANA Response Live documentation that is listed in “Related Documentation Resources” on [page 363](#).

Each of these internal functions also calls the `AddMessage()` function (defined earlier in this file) to display informational or diagnostic messages.

```

/**** HB API Events handlers ****/
function HBCouldNotConnect( sReasonID )

```

```
{
    AddMessage("Can't connect to Cobrowse server. Reason: "+sReasonID);
}

function HBLoginError(reasonID, description)
{
    AddMessage("Can't login to Cobrowse server. Reason: " + reasonID + " Description: " +
        description);
}

function HBJoinedSuccessfully()
{
    IsAgentJoined = true;
    AddMessage ("User joined.");
}

function HBJoinRequested(sName)
{
    AddMessage ("Event HBJoinRequested(" + sName + ")");
    return true;
}

function HBSessionEnded()
{
    AddMessage("Conavigation session has ended.");
}

function HBLoggedIn(sHipboneID)
{
    HBUserID = sHipboneID;
    AddMessage ("CobrowseID : " + sHipboneID + ".");
    UserLoggedIn = true;
    HBApiWindow.HBCreateSession();
}

function HBLoggedOut(sReasonID)
{
    AddMessage("You have been logged out.");
    UserLoggedIn = false;
    HBUserID = "";
}

function HBSessionStarted()
{
    AddMessage ("HBSessionStarted()");
    HBApiWindow.HBConavigateLink(StartPage, null);
}

function HBLinkConavigated(sLink, sTarget, sUserName)
{
    AddMessage ("Event HBLinkConavigated: " + sLink);
}
```

```
}

function HBUserExitedSession(name)
{
    AddMessage ("Event HBUserExitedSession(" + name + ")");
    if (HBUserID == name)
        doExitSession();
}

function HBUserEnteredSession(name)
{
    AddMessage ("User with ID : " + name + " has joined to session.");
    //HBApiWindow.HBReload("HB_HIPBONE"); // fix for join bug
}...
```

Chat and Cobrowse Sample

The .../ChatAndCoBroswe directory contains the files for the Chat and Cobrowse Sample.

Purpose

The Chat and Cobrowse Sample demonstrates how to use cobrowsing functionality during a chat session.

Functionality Overview

The following sections outline the code that used for implementing the Chat and Cobrowse Sample:

- “Differences Between Common Files” on [page 204](#)
- “Getting Instances of Load Balancer and CoBrowse Server” on [page 205](#)
- “Declarations” on [page 205](#)
- “Functions” on [page 206](#)
- “Event Handlers” on [page 208](#)
- “Drawing the Chat Form and Cobrowse Links” on [page 210](#)

For information about code that is common to all of the samples—such as retrieving parameters, setting the content type, character encoding, loading libraries, and importing files—see “Common Code” on [page 107](#).

Files

The .../ChatAndCoBroswe directory consists of 12 files. Seven of these are common to all of the cobrowse samples and are described in “Cobrowse Samples Overview” on [page 197](#):

- blank.html
- hbapi.html
- hbmessage_to_var.html
- hbmessagingform.html
- hbmessage_to_var.js
- hbmessaging.js
- qstring.js

Three files are shared with the Chat Sample and are detailed in “Chat Sample: Files” on [page 122](#):

- HtmlChatFrameSet.jsp—Virtually identical to its Chat Sample counterpart.
- HtmlChatCommand.jsp—Virtually identical to its Chat Sample counterpart.
- HtmlChatPanel.jsp—Contains added code (compared to its Chat Sample counterpart) that provides cobrowsing functionality. See the detailed code explanation in the next section.

One file is common to the Advanced Chat Sample and is explained in detail in “Advanced Chat” on [page 132](#):

- ChatTranscript.jsp—Virtually identical to its Advance Chat Sample counterpart.

The last file is the main frameset of this sample:

- ChatAndCoBrowse.htm

Code Explanation

The following subsections explain the code that is in the Chat and Cobrowse Sample.

Differences Between Common Files

This section highlights the ways in which the Chat and Cobrowse Sample’s HtmlChatPanel.jsp file differs from the two sample files from which it is basically constructed:

- The similarly named HtmlChatPanel.jsp file in “Chat Sample” on [page 121](#). The two files share certain chat functions, as well as the HTML code that draws a chat form.
- The CoBrowseEventHandler.jsp file in “Basic Cobrowse Sample” on [page 197](#). The HtmlChatPanel.jsp file of this sample adds cobrowsing functions and event handlers from that file. However, some event handlers differ in ways that this section also identifies.

First, comparing the HtmlChatPanel.jsp file of this sample to the HtmlChatPanel.jsp file of the Chat Sample: The file of this sample imports two

packages that are instead handled by the `HtmlChatCommand.jsp` file
Chat Sample:

```
<%@ page import="com.genesyslab.webapi.core.LoadBalancer" %>
<%@ page
import="com.genesyslab.platform.configuration.protocol.types.CfgApp
Type" %>
```

Getting Instances of Load Balancer and CoBrowse Server

The following try/catch block is also absent from the `HtmlChatPanel.jsp` file of the Chat Sample; instead it corresponds to the `CoBrowseEventHandler.jsp` file of the Cobrowse Sample. As in that sample, it creates a Load Balancer instance and attempts to discover a Cobrowse Server host:

```
String CoBrowseServerHost = null;
try
{
    ServiceInfo si = LoadBalancer.GetServiceInfo(CfgAppType.CFGCoBrowsingServer,
        strTenant);
    if (si != null)
        CoBrowseServerHost = si.getHost();
}
catch(Exception ex)
{
    //Error load balancing is disabled.
    CoBrowseServerHost = null;
}
```

If Load Balancer does not provide a Cobrowse Server, you obtain a server from the `constants.jsp` file:

```
if(CoBrowseServerHost != null)
    out.println("CobrowseHostName = \"" + CoBrowseServerHost + "\";"); //From
    Loadbalancer
else
    out.println("CobrowseHostName = \"" + strConavServerUrl + "\";"); //From
    constants.jsp file

out.println("var ConavigationiChannelID = \"" + strConavChannelID + "\""); //From
    constants.jsp file
```

Declarations

The body section of the `HtmlChatPanel.jsp` file omits the corresponding reference to `CommLib.js` of the Chat Sample file. However, it declares and initializes some extra variables to support cobrowsing functions. Again, these

declarations match those that are in the `CoBrowseEventHandler.jsp` file of the Cobrowse Sample:

```
var HBApiWindow = null;
var HBUserID = "";
var IsFirstConavigation = true;

var UserLoggedIn = false;
var ConnectTo = "";
var StartPage = "http://www.google.com";
```

Functions

As in the `HtmlChatPanel.jsp` file of the Chat Sample, this file includes a `show_message()` function. Elsewhere in this file, cobrowsing event handlers will call the `show_message()` function to write messages to the chat frame:

```
function show_message(strNickName, strMessage, iUserType)
{
    //iUserType == 0 from Agent
    //iUserType == 1 from Client
    //iUserType == 2 from External
    //iUserType == 3 from Supervisor
    //iUserType == 4 from System
    //iUserType == 5 command from System to push URL from agent
    if (iUserType == 0)
    {
        window.frames.ChatTranscript.AddMessage (strNickName + ":", AgentNickNameColor,
1, 1);
        window.frames.ChatTranscript.AddMessage (strMessage, AgentMessageColor, 1, 0);
        hide_agent_typing();
    }
    else if (iUserType == 1)
    {
        window.frames.ChatTranscript.AddMessage (strNickName + ":", ClientNickNameColor,
1, 1);
        window.frames.ChatTranscript.AddMessage (strMessage, ClientMessageColor, 1, 0);
    }
    else if (iUserType == 2 || iUserType == 3)
    {
        window.frames.ChatTranscript.AddMessage (strNickName + ":", AgentNickNameColor,
1, 1);
        window.frames.ChatTranscript.AddMessage (strMessage, AgentMessageColor, 1, 0);
    }
    else if (iUserType == 4)
    {
        if (strMessage == "Agent is typing.")
        {
            show_agent_typing(strNickName);
        }
    }
}
```

```

        else if (strMessage == "Agent has stopped typing.")
        {
            hide_agent_typing();
        }
        else
        {
            window.frames.ChatTranscript.AddMessage("System:", ActionColor, 1, 1);
            window.frames.ChatTranscript.AddMessage(strMessage, ActionColor, 1, 0);
        }
    }
    else if (iUserType == 5)
    {
        hide_agent_typing();
        var bNeedToShowMessage = false;
        var winNewWindow = window.open (strMessage, "PushUrlWindow"+iOpenWindowCounter,
        "", false);
        iOpenWindowCounter++;

        if (winNewWindow == null)
            bNeedToShowMessage = true;
        else
        {
            if (window.opera)
            {
                if (!winNewWindow.opera)
                    bNeedToShowMessage = true;
            }
        }

        if (bNeedToShowMessage)
        {
            window.frames.ChatTranscript.AddMessage ("System: pop-up blocker detected.
            ", AgentNickNameColor, 1, 1);
            window.frames.ChatTranscript.AddMessage ("Please navigate to the next link:
            " + strMessage, AgentMessageColor, 1, 0);
        }
    }
}

```

As in the `CoBrowseEventHandler.jsp` file of the Cobrowse Sample, this version of `HtmlChatPanel.jsp` includes additional logic, functions, and event handlers to support interactions with the Cobrowsing Server.

```

function CoBrowse_onclick()
{
    if(UserLoggedIn == true)
    {
        alert("Please log out user "+HBUserID+" before starting a new co-browse
        session.");
    }
}

```

```

else
{
    show_system_message("Connecting to Cobrowse server...");
    if (HBApiWindow == null)
    {
        HBApiWindow= parent.parent.hbapi;
        var strUrl= new String(window.location.href);
        var strProcessorUrl= strUrl.substring(0,
strUrl.lastIndexOf("/")+1)/hbmessage_to_var.html";
        HBApiWindow.HBInitializeAPI("main", CobrowseHostName, strProcessorUrl);
        HBApiWindow.HBLoginGuest(ConavigationiChannelID, "", ConnectTo, false,
"acctSpecificData");
    }
    else
    {
        HBApiWindow.HBLoginGuest(ConavigationiChannelID, "", ConnectTo, false);
    }
}
}

```

Event Handlers

The first three event handlers are similar to identically named event handlers in the `CoBrowseEventHandler.jsp` file of the Cobrowse Sample. However, whereas the event handlers of that file call an `AddMessage()` function, the event handlers of this file instead call its `show_message()` function (see “Functions” on [page 206](#)), so as to write their messages to the chat frame. Note that these messages are not sent to the agent and are not visible in a chat transcript:

```

/**** HB API Events handlers ****/
function HBCouldNotConnect( sReasonID )
{
    show_system_message("Can't connect to Cobrowse server. Reason:
"+sReasonID);
}

function HBLoginError(reasonID, description)
{
    show_system_message("Can't login to Cobrowse server. Reason: " +
reasonID + " Description: " + description);
}

function HBJoinedSuccessfully()
{
    IsAgentJoined = true;
    show_system_message ("Agent joined.");
}

```

The `HBJoinRequested()` event handler of this file differs in the same way from its counterpart in the Cobrowse Sample. For example, instead of calling the `AddMessage()` function it calls the `show_message()` function to identify the agent to the caller:

```
function HBJoinRequested(sName)
{
    show_system_message ("Your agent is : " + sName + ".");
    return true;
}

function HBSessionEnded()
{
    show_system_message("Conaviagation session has ended.");
}
```

The `HBLoggedIn()` event handler of this file differs in two ways from its counterpart in the Cobrowse Sample. First, it calls the `show_message()` function instead of the `AddMessage()` function. Second, upon verifying the login and connection of the customer, it writes the cobrowse ID of the customer to the chat frame before it calls the `HBCreateSession()` function:

```
function HBLoggedIn(sHipboneID)
{
    HBUserID = sHipboneID;
    show_system_message ("CobrowseID : " + sHipboneID + ".");
    UserLoggedIn = true; ...
```

Specifically, this next code block sends the encoded message to the agent via chat. This way, the agent will know where to connect to begin cobrowsing with the customer:

```
...if (bConnected == true)
{
    HBSendCobrowseID();
}
HBApiWindow.HBCreateSession();
}
```

Finally, each of the remaining event handlers of this file differs from its counterpart of the Cobrowse Sample by internally calling the `show_system_message()` function instead of the `AddMessage()` function:

- `HBLoggedOut()`
- `HBSessionStarted()`
- `HBLinkConavigated()`
- `HBUserExitedSession()`
- `HBUserEnteredSession()`

Drawing the Chat Form and Cobrowse Links

Along with the HTML code that creates the traditional chat form, the code displays the host name of the Cobrowsing Server, and displays links for starting and stopping a cobrowse session:

```
<form name="chat_form" id=chat_form onSubmit="javascript:on_send(); return false;">
<table style="width:100%" cellpadding="0" cellspacing="2">
  <tr>
    <td class="samplesSampleTopToolbar">
      <table>
        <tr>
          <td style="width:253px" class="samplesSampleNameTitleWhite">Chat and
Cobrowse Sample</td>
          <td style="width:91px;font:8pt verdana;"
onClick="javascript:on_connect();" class="sampleGreenButtonBig"
onmouseover="javascript:MouseOver(this);"
onmouseout="javascript:MouseOut(this);">Start chat</td>
          <td style="width:91px;font:8pt verdana;"
onClick="javascript:on_disconnect();" class="sampleRedButtonBig"
onmouseover="javascript:MouseOver(this);"
onmouseout="javascript:MouseOut(this);">Stop chat</td>
          <td style="width:91px;font:8pt verdana;"
onClick="javascript:CoBrowse_onclick();" class="sampleGreenButtonBig"
onmouseover="javascript:MouseOver(this);"
onmouseout="javascript:MouseOut(this);">Start cobrowse</td>
          <td style="width:91px;font:8pt verdana;"
onClick="javascript:EndCoBrowse_onclick();" class="sampleRedButtonBig"
onmouseover="javascript:MouseOver(this);"
onmouseout="javascript:MouseOut(this);">Stop cobrowse</td>
          <td style="width:auto" ></td>
        </tr>
      </table>
    </td>
  </tr>
  <tr>
    <td class="samplesSampleParagraph">Fill in your personal information in fields
below to proceed.</td>
  </tr>
  <tr>
    <td>
      <table style="width:617px" border="0">
        <tr>
          <td class="samplesSampleFieldTitle" ></td>
          <td></td>
          <td style="width:6px; vertical-align: middle;"></td>
        </tr>
      </table>
    </td>
  </tr>
</table>
</form>
```

```

        <tr>
            <td class="samplesSampleFieldTitle" >First name:</td>
            <td><input class="samplesFieldStyle" type="text" name="FirstName"
value="" /></td>
            <td style="width:6px; vertical-align: middle;"><span
class="samplesMandatoryMark">*</span></td>
        </tr>...

```

Cobrowse with Meet Me

The `.../CoBrowseMeetMe` directory contains the files for the Cobrowse with Meet Me Sample. This sample demonstrates cobrowse-with-meet-me functionality.

Purpose

The Cobrowse with Meet Me Sample code demonstrates how to set up a cobrowse session with a specific other person whose cobrowse ID the user knows.

Functionality Overview

The following sections outline the code used to implement the Cobrowse with Meet Me Sample:

- “Entering the Cobrowse ID of the Other Party” on [page 212](#)
- ““Meeting” the Other Party” on [page 212](#)
- “Event Handlers” on [page 213](#)

For information about code that is common to all of the samples—such as retrieving parameters, setting the content type, character encoding, loading libraries, and importing files—see “Common Code” on [page 107](#).

Files

The `.../CoBrowseMeetMe` directory contains nine files; eight of these are documented in “Cobrowse Samples Overview” on [page 197](#). This section focuses on code that is unique to `CoBrowseEventHandler.jsp` file of this sample (as compared to the corresponding file in “Basic Cobrowse Sample” on [page 197](#)).

Code Explanation

The following subsections explain the code that is in the Cobrowse with Meet Me Sample.

Entering the Cobrowse ID of the Other Party

Compared to the `CoBrowseEventHandler.jsp` file of the Cobrowse Sample, this sample adds an input box and a link by which the user can specify the cobrowse ID of the person with whom the user wants to browse with:

```
<tr >
    <td style="width:507px"
class="samplesSampleNameTitleWhite">Cobrowse &quot;Meet Me&quot; Sample</td>
    <td style="width:91px"
onclick="javascript:CoBrowse_onclick();" class="sampleGreenButtonBig"
onmouseover="javascript:MouseOver(this);"
onmouseout="javascript:MouseOut(this);">Start</td>
    <td style="width:91px"
onclick="javascript:EndCoBrowse_onclick();" class="sampleRedButtonBig"
onmouseover="javascript:MouseOver(this);"
onmouseout="javascript:MouseOut(this);">Stop</td>
    <td style="width:auto" ></td>
</tr>
</table>
</td>
</tr>
<tr>
    <td class="samplesSampleParagraph">Cobrowse functionality sample. Also
allows to connect to another user by known Cobrowse ID of this user.</td>
</tr>

<tr>
    <td>
    <table border="0">
    <tr>
        <td class="samplesSampleFieldTitle" ></td>
        <td></td>
        <td style="width:6px; vertical-align: middle;"></td>
    </tr>

<tr>
        <td class="samplesSampleFieldTitle" >Connect to:</td>
        <td><input class="samplesFieldStyle" type="text" id="ConnectTo"
value="" name="ConnectTo"/></td>
        <td style="width:6px; vertical-align: middle;"><font
class="samplesMandatoryMark">*</font></td>
    </tr>...
```

“Meeting” the Other Party

The preceding code captures a `ConnectTo` value, which represents the target cobrowse ID. In the following code, the `CoBrowse_onclick()` function of this

sample includes additional logic that attempts to connect to this ID and notifies the user if the ConnectTo value is empty:

```
function CoBrowse_onclick()
{
    ConnectTo = trim(document.forms[0].ConnectTo.value);

    if(UserLoggedIn == true)
    {
        alert("Please log out user "+HBUserID+" before starting a new co-browse session.");
    }
    else if(ConnectTo == "")
    {
        alert("Please enter customer id you're trying to connect to");
    }
    else
    {
        AddMessage("Connecting to Cobrowse server...");
        if (HBApiWindow == null)
        {
            HBApiWindow= parent.hbapi;
            var strUrl= new String(window.location.href);
            var strProcessorUrl= strUrl.substring(0,
strUrl.lastIndexOf("/")+"/"+hbmessage_to_var.html";
```

Notice in the following code, that the ConnectTo variable is passed into both the HBInitializeAPI method and the HBLoginGuest method.

```
HBApiWindow.HBInitializeAPI("EventHandlerFrame", CobrowseHostName,
    strProcessorUrl);
HBApiWindow.HBLoginGuest(ConavigationiChannelID, "", ConnectTo, false,
    "acctSpecificData");
}
else
{
    HBApiWindow.HBLoginGuest(ConavigationiChannelID, "", ConnectTo, false);
}
AddMessage("Connecting to Cobrowse server...");...
```

Event Handlers

Most of the event handlers of this file correspond to those that are in the CoBrowseEventHandler.jsp file of the Basic Cobrowse Sample. However, there are minor differences.

The HBLoggedIn() event handler of this file omits the internal call to HBApiWindow.HBCreateSession() that is found in the HBLoggedIn() event handler of the Basic Cobrowse Sample, because you can assume that the other party has already created the session:

```
function HBLoggedIn(sHipboneID)
{
    HBUserID = sHipboneID;
    AddMessage ("CobrowseID : " + sHipboneID + ".");
    UserLoggedIn = true;
}
```

Similarly, the `HBSessionStarted()` event handler of this file differs from its counterpart in the Cobrowse Sample by omitting an internal call to `HBApiWindow.HBConavigateLink(StartPage, null)`. Again, you can assume that the other party has already established the URL to cobrowse:

```
function HBSessionStarted()
{
    AddMessage ("HBSessionStarted()");
}
```

For similar reasons, this file's `HBUserEnteredSession()` event handler omits its Cobrowse Sample counterpart's internal call to `HBApiWindow.HBConavigateLink(StartPage, null)`:

```
function HBUserEnteredSession(name)
{
    AddMessage ("User with ID : " + name + " has joined to
        session.");
    //HBApiWindow.HBReload("HB_HIPBONE"); // fix for join bug
}
```

Cobrowse with Initial Startup Page

The `.../CoBrowseInitStartupPage` directory contains the files for the Cobrowse with Initial Startup Page Sample.

Purpose

The code for the Cobrowse with Initial Startup Page Sample demonstrates basic cobrowsing that uses a specified initial startup page.

Functionality Overview

The following sections outline the code that is used to implement the Cobrowse with Initial Startup Page Sample:

- “Specifying the Startup Page” on [page 215](#)
- “Event Handlers” on [page 217](#)

For information about code that is common to all of the samples—such as retrieving parameters, setting the content type, character encoding, loading libraries, and importing files—see “Common Code” on [page 107](#).

Files

This section focuses on code that is unique to this sample, as compared to the basic “Basic Cobrowse Sample” on [page 197](#). The `.../CoBrowseInitStartupPage` directory contains nine files; seven of these are common files that already have been described in “Cobrowse Samples Overview” on [page 197](#). Two files differentiate this sample:

- `InitialStartupPageExample.html`—Replaces the `CoBrowse.htm` file of the Basic Cobrowse Sample, but serves a similar function: Sets up the basic display frame as a container for imported logic.
- `CoBrowseEventHandler.jsp`—Contains additional logic, compared to the corresponding Cobrowse Sample file of the same name. For details, see the following section.

Code Explanation

The following subsections explain the code that is in the Cobrowse with Initial Startup Page Sample.

Specifying the Startup Page

Compared to its counterpart in the Basic Cobrowse Sample, the `CoBrowseEventHandler.jsp` file of this sample contains one additional code block that provides:

- An input box to specify the cobrowse ID of the other party.
- An input box to confirm or override the initial URL that both users will cobrowse
- A link to open that page.

The following shows the added code:

```
<tr>
    <td class="samplesSampleTopToolbar">
        <table >
            <tr >
                <td style="width:511px"
class="samplesSampleNameTitleWhite">Cobrowse Start Page Sample</td>
                <td style="width:91px"
onClick="javascript:CoBrowse_onclick();" class="sampleGreenButtonBig"
onmouseover="javascript:MouseOver(this);"
onmouseout="javascript:MouseOut(this);">Start</td>
```

```

        <td style="width:91px"
onclick="javascript:EndCoBrowse_onclick();" class="sampleRedButtonBig"
onmouseover="javascript:MouseOver(this); "
onmouseout="javascript:MouseOut(this);">Stop</td>
        <td style="width:auto" ></td>
    </tr>
</table>
</td>
</tr>
<tr>
    <td class="samplesSampleParagraph">Cobrowse functionality sample which
also allows start session from particular page.</td>
</tr>

<tr>
    <td>
        <table border="0">
            <tr>
                <td class="samplesSampleFieldTitle" ></td>
                <td></td>
                <td style="width:6px; vertical-align: middle;"></td>
            </tr>

            <tr>
                <td class="samplesSampleFieldTitle" >Connect to:</td>
                <td><input class="samplesFieldStyle" type="text" id="ConnectTo"
value="" name="ConnectTo"/></td>
                <td style="width:6px; vertical-align: middle;">&nbsp;</td>
            </tr>
            <tr>
                <td class="samplesSampleFieldTitle" >Initial startup page:</td>
                <td><input class="samplesFieldStyle" type="text" id="StartPage"
value="http://www.yahoo.com" name="StartPage"/></td>
                <td style="width:6px; vertical-align: middle;"><font
class="samplesMandatoryMark">*</font></td>
            </tr>

            <tr>
                <td class="samplesSampleFieldTitle" >Events:</td>
                <td><textarea style="height:200px" class="samplesFieldStyle"
rows="10" cols="60" id="Messages" name="Messages"
readonly="readonly"></textarea></td>
                <td style="width:6px; vertical-align: middle;"></td>
            </tr>
        </table>
    </td>
</tr>

```

Event Handlers

Most of the event handlers of this file correspond to those event handlers in the `CoBrowseEventHandler.jsp` file of the Basic Cobrowse Sample. This section identifies the minor differences.

In the version of the `HBLoggedIn()` event handler of this file, the internal call to `HBApiWindow.HBCreateSession()` is embedded in an `if` branch. This `if` branch verifies a connection to the other party before it creates a session:

```
function CoBrowse_onclick()
{
    if(UserLoggedIn == true)
    {
        alert("Please log out user "+HBUserID+" before starting a new
cobrowse session.");
    }
    else
    {
        ConnectTo = document.forms[0].ConnectTo.value;
        StartPage = document.forms[0].StartPage.value;
        if (HBApiWindow == null)
        {
            HBApiWindow= parent.hbapi;
            var strUrl= new String(window.location.href);
            var strProcessorUrl= strUrl.substring(0,
                strUrl.lastIndexOf("/")+1)/hbmessage_to_var.html";
            HBApiWindow.HBInitializeAPI("EventHandlerFrame",
                CobrowseHostName, strProcessorUrl);
            HBApiWindow.HBLoginGuest(ConavigationiChannelID, "",
                ConnectTo, false, "acctSpecificData");
        }
        else
        {
            HBApiWindow.HBLoginGuest(ConavigationiChannelID, "",
                ConnectTo, false);
        }
        AddMessage("Connecting to Cobrowse server...");...
```

The `HBSessionStarted()` event handler of this file differs from its Basic Cobrowse Sample counterpart by omitting an internal call to `HBApiWindow.HBConavigateLink(StartPage, null)`. You can assume that the other party has already established both a cobrowse session and the URL to cobrowse:

```
function HBSessionStarted()
{
    AddMessage ("HBSessionStarted()");
}
```

The `HBApiWindow.HBConavigateLink(StartPage, null)` call instead occurs in the `HBUUserEnteredSession()` event handler of this file. (The Basic Cobrowse Sample version of that event handler makes no such call.) The call is embedded in an `if` branch that first verifies the cobrowse ID of the other party. It co-navigates to a start page only if the party has newly entered the session:

```
function HBUUserEnteredSession(name)
{
    AddMessage ("User with ID : " + name + " has joined to
        session.");
    //HBApiWindow.HBReload("HB_HIPBONE"); // fix for join bug
    if(HBUUserID == name)
    {
        HBApiWindow.HBConavigateLink(StartPage, null);
    } ...
}
```

Cobrowse with Dynamic Startup Page Sample

The `.../CoBrosweDynamicStartPage` directory contains the Cobrowse with Dynamic Startup Page Sample.

Purpose

The Cobrowse with Dynamic Startup Page Sample demonstrates how one party can click a cobrowse control on an HTML page that contains a form. When cobrowse begins, all of the form data that already is entered is dynamically prefilled in the cobrowse window of the other party, so that a customer and an agent can access the form data of the other dynamically.

Functionality Overview

The following sections outline the code that is used to implement the Cobrowse with Dynamic Startup Page Sample:

- “Functions” on [page 219](#)
- “Getting User Data” on [page 219](#)

For information about code that is common to all of the samples—such as retrieving parameters, setting the content type, character encoding, loading libraries, and importing files—see “Common Code” on [page 107](#).

Files

The `.../CoBrosweDynamicStartPage` directory contains five files. In order for the sample to work properly, place all the files together into a public web directory.

Four of these files (listed in “Cobrowse with Dynamic Startup Page” on [page 56](#)) should not be modified.

This section focuses on the file that you can modify—the sample’s main file; `ExampleOfDynamicStartupPage.jsp`. This file serves some of the same purposes as the `CoBrowseEventHandler.jsp` file in “Basic Cobrowse Sample” on [page 197](#); therefore, this discussion focuses on the unique code that it provides.

The cobrowse session starts from the `ExampleOfDynamicStartupPage.jsp` page, where it collects user input. To check the sample, enter arbitrary information in the form fields of this page, then click the `Live Help` link. You will see a cobrowse window that is specific to the dynamic-startup page.

Code Explanation

The following subsections explain the code that is in the Cobrowse with Dynamic Startup Page Sample.

Functions

The sample code defines an `openLiveHelp()` function. This function opens the Dynamic Startup Page API window when the user clicks the `Live Help` link:

```
function openLiveHelp()
{
    startDSPMeetMe(ConavigationiChannelID, null, "guest", null,
        CobrowseHostName)
}
```

Getting User Data

The file then builds the form that solicits the input of personal information by the user. Two text boxes prompt for the name and e-mail address of the user:

```
<tr>
    <td class="samplesSampleFieldTitle" >Full Name:</td>
    <td><input style="width:550px; vertical-align: middle;" type="text"
class="samplesFieldStyle" value="" name="fullName"/></td>
    <td style="width:6px; vertical-align: middle;"><font
class="samplesMandatoryMark">*</font></td>
</tr>

<tr>
    <td class="samplesSampleFieldTitle" >Email:</td>
    <td><input style="width:550px; vertical-align: middle;" type="text"
class="samplesFieldStyle" value="" name="emailAddress"/></td>
    <td style="width:6px; vertical-align: middle;"><font
class="samplesMandatoryMark">*</font></td>
</tr>
```

A very basic drop-down list offers three State/Province options; then another text box prompts for the password of the user:

```
<tr>
    <td class="samplesSampleFieldTitle" >State</td>
    <td>
        <select style="width:550px; vertical-align: middle; "
class="samplesFieldStyle" name="state" id="state">
            <option value="">State/Province</option>
            <option value="CA">CA</option>
            <option value="NY">NY</option>
            <option value="TX">TX</option>
        </select>
    </td>
    <td style="width:6px; vertical-align: middle;"><font
class="samplesMandatoryMark">*</font></td>
</tr>

<tr>
    <td class="samplesSampleFieldTitle" >Password:</td>
    <td><input style="width:550px; vertical-align: middle; "
class="samplesFieldStyle" type="password" value="" name="password"/></td>
    <td style="width:6px; vertical-align: middle;"><font
class="samplesMandatoryMark">*</font></td>
</tr>
```

Next, a radio-button panel prompts the user to select a credit-card type:

```
<tr>
    <td class="samplesSampleFieldTitle" >Credit card type:</td>
    <td style="width:550px; vertical-align: middle;">
        <table style="width:100%" class="samplesFieldStyle" border="0">
            <tr>
                <td style="max-width:10px; vertical-align: middle;">
                    <input type="radio" value="Visa" checked="checked"
name="creditCardType" id="creditCardType_0" />
                </td>
                <td>
                    Visa
                </td>
                <td style="max-width:10px; vertical-align: middle;">
                    <input type="radio" value="MasterCard" name="creditCardType"
id="creditCardType_1" />
                </td>
                <td>
                    Mastercard
                </td>
                <td style="max-width:10px; vertical-align: middle;">
                    <input type="radio" value="AmericanExpress"
name="creditCardType" id="creditCardType_2" />
                </td>
                <td>
                    American Express
                </td>
            </tr>
        </table>
    </td>
</tr>
```

```

</td>
<td>...

```

A multiple-selection box then allows the user to specify one or more preferred callback times:

```

<tr>
    <td class="samplesSampleFieldTitle" >Preferred callback time:</td>
    <td style="width:550px; vertical-align: middle;">
        <select style="width:100%; height:50px; vertical-align: middle;"
            class="samplesFieldStyle" multiple="multiple" name="selectbox1" size="3">
            <option value="Morning">Morning</option>
            <option value="Day time">Day time</option>
            <option value="Evening">Evening</option>
        </select>
    </td>...

```

Finally, the code defines a link to the Live Help functionality, which is provided by the other files in the installed directory of the sample:

```

<tr>
    <td colspan="3" class="samplesFormBtns">
        <input type="hidden" name="action" value="">
        <ul class="samplesUL">
            <li class="samplesMenuItem">
                <a href="javascript:openLiveHelp()">Live Help</a>
            </li>
        </ul>...

```

Build Your Own Dynamic Startup Page Example

To add Dynamic Startup Page functionality to your own custom page, first import the following libraries:

```

<%@ page import="Genesys.webapi.system.loadbalancing.*" %>
<%@ page import="Genesys.CfgLib.*" %>

```

Next, load the following JavaScript file, which provides the cobrowse API:

```

<script LANGUAGE=javascript type="text/javascript" SRC="responseLive.js"></script>

```

Then insert the following script:

```

String CoBrowseServerHost = null;
try
{
    ServiceInfo si = LoadBalancer.GetServiceInfo(CfgAppType.CFGCoBrowsingServer,

```

```

        strTenant);
    if (si != null)
        CoBrowseServerHost = si.getHost();
}
catch(Exception ex)
{
    //Error load balancing is disabled.
    CoBrowseServerHost = null;
}...

if(CoBrowseServerHost != null)
    //From Loadbalancer
    out.println("CobrowseHostName = \"\" + CoBrowseServerHost + "\"");
else
    //From constants.jsp
    out.println("CobrowseHostName = \"\" + strConavServerUrl + "\"");

    //From constants.jsp
    out.println("var ConavigationiChannelID = \"\" + strConavChannelID + "\"");
...
function openLiveHelp()
{
    startDSPMeetMe(ConavigationiChannelID,null,"guest",null, CobrowseHostName);
}...

```

To each page that will incorporate Dynamic Startup Page functionality, add a reference that starts the `openLiveHelp()` function:

```
<a href="javascript:openLiveHelp()">Live Help</a>
```

Settings

To ensure that your Dynamic Startup Page application will work, observe the following recommended settings:

- If you are using a test certificate for the HTTPS connection to your dynamically browsed page, you must place the certificate-authority file into the `.../hroot/certs/` folder of the Cobrowse Server.
- The dynamic startup page uses cookies to transfer information about the page. Therefore, users must enable cookies in their browser settings. Your application should warn users about this requirement and, if possible, run a diagnostic test for whether cookies can be set in the browser of the user.

Limitations of the Dynamic Startup Page

- **IP checks**—The dynamic startup page does not work if your website associates identification and cookies with the IP address of the computer of the customer. This is a very unusual circumstance, and usually means that your site does not work with AOL browsers or certain corporate firewalls.

- **POST submissions**—You must perform special custom work to enable the startup page to reappear after a POST submission.
- **Cookies**—The dynamic startup page feature traps only the cookies that are accessible from the domain and path in which the Help button is located.
- **Large cookies**—The dynamic startup-page feature cannot support cookies that are greater than 1KB in size.
- **Frames in different domains**—Pages that have frames that use different domains or protocols must be supported on a custom basis.
- **Startup pages within frames**—Not supported.
- **Multiple submissions**—If your website does not reload URLs, the startup page feature might not function properly.



Chapter

10

eServices Samples for .NET

This chapter examines eServices's web-based samples for .NET, and their code. (Java developers should instead see Chapter 9, "eServices Samples for Java," on [page 101](#).) This chapter covers the following topics:

- [Overview, page 225](#)
- [Shared Files, page 228](#)
- [Web Callback Sample, page 230](#)
- [Chat Sample, page 238](#)
- [Chat with AJAX Sample, page 251](#)
- [Survey Sample, page 262](#)
- [E-Mail Sample, page 269](#)
- [Genesys 3rd Party Media Sample, page 274](#)
- [Stat Server Sample, page 286](#)
- [Universal Contact Server Sample, page 294](#)
- [Cobrowse Samples Overview, page 308](#)
- [Basic Cobrowse Sample, page 309](#)
- [Chat and Cobrowse Sample, page 315](#)
- [Cobrowse with Meet Me, page 322](#)
- [Cobrowse with Initial Startup Page, page 326](#)
- [Cobrowse with Dynamic Startup Page Sample, page 330](#)

Note: The font size for the code in this chapter has been reduced to display long code lines.

Overview

This chapter explains how to implement chat, e-mail, statistics, history, and 3rd Party Media submission functions in your web application, by reviewing the

key functions in the respective samples. The samples come with the eServices Interactive Management CD. For information on installing the samples, see Chapter 2, “About the Samples,” on [page 45](#).

Please note the following about the sample code presented and organized in this chapter:

- The code is excerpted from the actual sample code, as the actual code is too long to be displayed here.
- The excerpted code illustrates a point, or calls attention to a particular feature. You should refer to the actual code and to the appropriate API reference(s) for further information.
- Although this chapter reviews the different functions that each sample performs, some of these functions, and the excerpted code, may not be presented in the same order or layout as in the sample.

Note: The `LoadBalancer.GetServiceInfo(ConfServerClientType, String)` method has been deprecated. It has been replaced by the `LoadBalancer.GetServiceInfo(CfgAppType, String)` method.

Samples Included

The samples discussed here are:

- Web Callback
- Web-Based Chat
- Web-Based Chat with AJAX
- Web-Based Survey
- Web-Based E-mail
- Web-Based Genesys 3rd Party Media Interaction Submission
- Web-Based Stat Server
- Web-Based Universal Contact Server
- Web-Based Basic Cobrowse
- Web-Based Chat and Cobrowse
- Web-Based Cobrowse with Meet Me
- Web-Based Cobrowse with Initial Startup Page
- Web-Based Cobrowse with Dynamic Startup Page

The .NET samples that are not discussed here include:

- Advanced Chat
- Chat Widget
- Dynamic Invitation

Please refer to Chapter 9, “eServices Samples for Java,” on [page 101](#) for information about these samples.

[Figure 32](#) shows the SimpleSamples802 directory structure of the .NET version of the samples.

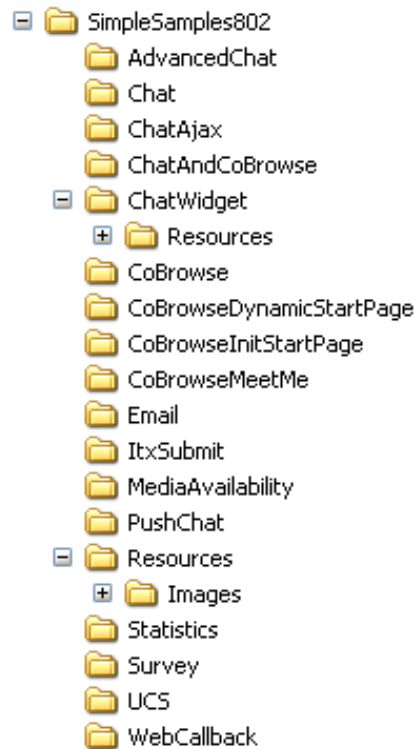


Figure 32: Samples .NET Directory Structure

Web-Based Chat

This sample demonstrates how to implement a chat feature on a web form.

Web-Based Chat with AJAX

This sample demonstrates how to implement a chat feature on a web form.

Web-Based Survey

This sample demonstrates how to implement a survey in a chat form.

Web-Based E-Mail

This sample demonstrates how to send e-mail using a web form.

Web-Based Genesys 3rd Party Media

This sample demonstrates how to submit and update an interaction using a web form.

Web-Based Stat Server

This sample demonstrates how to select predefined statistics and retrieve their current value using a web form.

Web-Based Universal Contact Server

This sample demonstrates how to access a contact's interaction history using a web form.

Files Included: .ASPX Versus .ASPX.CS

Each sample includes at least one pair of files named and organized according to the following pattern:

- `<SampleName>.aspx.cs`: C# source file that provides core package inheritance statements, declarations, and functions. The sample descriptions in this chapter focus on these C# files.
- `<SampleName>.aspx`: Controls the presentation of the C# code on a generated server page. Defines the basic form design, including some JavaScript functions. Also provides certain directives for IIS (Internet Information Server), as in this example:

```
<%@ Page Language="C#" AutoEventWireup="true"
    CodeFile="Callback.aspx.cs" Inherits="Callback_Callback" %>
```

Shared Files

The files listed below are used in the web samples. The following subsections group the substantive files into categories and explain them in detail.

- `CommLib.js`—see “Common Functions” on [page 229](#).
- `App_Code\Constants.cs`—see “Constants” on [page 229](#).
- `Global.asax`—see “Startup, Shutdown, and Load Balancing” on [page 229](#).
- `icc_style.css`—see “Style Sheets” on [page 230](#).
- `index.htm`—see “Greetings Page” on [page 230](#).
- `ip_description.xml`—XML describing the installation package.
- `mcr_style.css`—see “Style Sheets” on [page 230](#).
- `read_me.html`—HTML describing the installation package.

- `Web.Config`—see “Licensing, Authentication, and Error Reporting” on [page 229](#).
- `arrow.gif`—see “Graphics” on [page 230](#).
- `fon.gif`—see “Graphics” on [page 230](#).
- `genesyslogo-trans.gif`—see “Graphics” on [page 230](#).
- `lms\webapiserverdotnet.lms`—provides log messages for these samples.

File Descriptions

Common Functions

The `CommLib.js` file stores common functions that all the samples use. It contains functions that:

- Identify a user’s web browser.
- Retrieve the handle to HTML frames or control objects.
- Perform basic string manipulation and encoding.
- Retrieve submitted form parameters and return the current time.

Constants

The `constants.cs` file defines one private and one public constant, which together identify the tenant:

```
private string strTenantName = "<YOUR_TENANT_NAME_HERE>";  
public string TenantName
```

Startup, Shutdown, and Load Balancing

The `Global.asax` file handles initiation and shutdown for the applications, sessions, server-side load balancing, and pertinent system tables and variables.

Licensing, Authentication, and Error Reporting

The `Web.Config` file inherits most of its contents from Microsoft’s .NET Framework, with the addition of Genesys licensing parameters. Other editable parameters here control ASP.NET’s authentication mode, and enable you to substitute custom error pages for generic HTML error pages (403, 404, and so on).

Presentation

Greetings Page

The `index.htm` file is the main or greeting page to access the web samples. The file presents an HTML table menu with links to each of the samples.

Graphics

`Arrow.gif`, `fon.gif`, and `genesyslogo-trans.gif` are graphics files used in `index.htm` and in various samples. The `Arrow.gif` graphic displays a forward arrowhead. The `fon.gif` graphic is tiled as background wallpaper. The `Genesyslogo-trans.gif` graphic displays the Genesys company logo.

Style Sheets

The cascading style sheet (CSS) file, `StyleSheet.css`, contains instructions for adding the bold font to header tags, adding tables, and other layout code. This guide does not discuss CSS technologies. You should be able to easily find CSS tutorials on the Web.

Web Callback Sample

Purpose

The Web Callback Sample is a file that shows how to:

- Create a web callback interaction
- Display a web callback interaction
- Display a list of web callback interactions
- Cancel a web callback interaction
- Reschedule a web callback interaction

Functionality Overview

The following sections review the code that is used in the implementation of the different callback functions:

- “Getting Load Balancer, Interaction Server, and Creating the Protocol Object” on [page 231](#)
- “Create the Web Callback Object” on [page 232](#)
- “Create a Web Callback Interaction” on [page 233](#)
- “Retrieve a Single Web Callback Interaction” on [page 234](#)
- “Retrieve a List of Web Callback Interactions” on [page 235](#)

- “Cancel a Web Callback Interaction” on [page 236](#)
- “Reschedule a Web Callback Interaction” on [page 237](#)

Files

The `.../WebCallback` directory contains the Web Callback Sample. The sample consists of two files, `WebCallback.aspx` and `WebCallback.aspx.cs`.

Code Explanation

The following subsections explain the code in the `WebCallback.aspx.cs` file. In some places several lines of code have been omitted in order to focus your attention on the more important points. In these cases, the missing lines have been replaced with “...”.

Getting Load Balancer, Interaction Server, and Creating the Protocol Object

This section of code creates an instance of Load Balancer and returns an available instance of Interaction Server. When you get an available Interaction Server, store the information about its alias in a hidden element and continue to use the same server every time:

```
private static InteractionServerProtocol is_protocol;

protected static InteractionServerProtocol ISProtocol
{
    get
    {
        try
        {
            if (is_protocol == null)
            {
                ServiceInfo si = LoadBalancer.GetServiceInfo
                    (CfgAppType.CFGInteractionServer, options.TenantName);

                Below, we create the protocol object, and set the client type and name before
                opening it:

                is_protocol = new InteractionServerProtocol(new Endpoint(new
                    Uri("tcp://"+si.Host+": "+si.Port.ToString())));
                is_protocol.ClientType = Genesyslab.Platform.OpenMedia.Protocols.
                    InteractionServer.InteractionClient.AgentApplication;
                is_protocol.ClientName = "WebCallback Sample";
                is_protocol.Closed += new EventHandler(is_protocol_Closed);
                is_protocol.Open();

                Next, we create an agent login request, and set the TenantId and PlaceId:

                RequestAgentLogin request = RequestAgentLogin.Create();
                request.TenantId =
```

```

        Convert.ToInt32(LoadBalancer.getTenantId(options.TenantName));
        request.PlaceId = options.WcbPlaceId;
        IMessage respondingEvent = is_protocol.Request(request);
        if(respondingEvent.Id == EventError.MessageId)
        {
            if(is_protocol != null)
            {
                is_protocol.Close();
            }
            throw new Exception(String.Format("Could not login to a place
                '{0}'", request.PlaceId));
        }
    }
}
}
catch (LoadBalancerException){
    if (is_protocol != null)
    {
        is_protocol.Close();
    }
    throw new Exception("Interaction Server is unavailable");
}
return is_protocol;...

```

Finally, we must close and dispose of the protocol object:

```

static void is_protocol_Closed(object sender, EventArgs e)
{
    is_protocol.Dispose();
    is_protocol = null;
}

```

Create the Web Callback Object

We create the WebCallback object, and set the TenantId and various queues:

```

private static WebCallback wc;

protected static WebCallback WcbProtocol
{
    get
    {
        try
        {
            if (wc == null)
            {
                wc = new WebCallback();
                wc.EnableLogging(LoadBalancer.GetLogger());
                wc.TenantId = Convert.ToInt32(LoadBalancer.getTenantId(options.TenantName));
                wc.NewQueue = options.WcbNewQueue;
                wc.CancelQueue = options.WcbCancelQueue;
                wc.ConferenceQueue = options.WcbConferenceQueue;
            }
        }
    }
}

```

```

        wc.TransferQueue = options.WcbTransferQueue;
        wc.RescheduleQueue = options.WcbRescheduleQueue;
    }
}
catch (LoadBalancerException)
{
    wc = null;
    return null;
}

return wc; ...

```

Create a Web Callback Interaction

The following code creates a web callback interaction using the `WCBRequestSubmit` method:

```

protected void LinkButtonRequestWebCallback_Click(object sender, EventArgs e)
{
    int startTime = Convert.ToInt32(CallbackTime.Text);
    int endTime = Convert.ToInt32(CallbackDuration.Text);

    if (FirstName.Text == string.Empty || LastName.Text == string.Empty
        || EmailAddress.Text == string.Empty || PhoneNumber.Text == string.Empty
        || Subject.Text == string.Empty)
    {
        ShowErrorMessage("Missing mandatory values");
        return;
    }
    else if (PhoneNumber.Text.Contains("") || PhoneNumber.Text.Contains("\\"))
    {
        ShowErrorMessage("Phone number (or IP) contains invalid characters");
        return;
    }
    WCBRequestSubmit request = WCBRequestSubmit.Create(PhoneNumber.Text);
    request.Subject = Subject.Text;
    request.DesiredResponseType = Media.Text; ...
}

```

Schedule a time for the web callback.

```

if (startTime == 0)
{
    request.Type = WebCallbackType.ASAP;
}
else
{
    request.Type = WebCallbackType.Scheduled;
    request.StartTime = DateTime.UtcNow.AddMinutes(startTime);
    request.EndTime = request.StartTime.AddMinutes(endTime);
    request.TimeShift = Convert.ToInt32(TimeShift.Value);
}

```

Set information regarding email notifications about the new callback interaction.

```
if (DoEmailNotifications.Text == "Yes")
{
    request.DoEmailNotification = true;
    request.NotificationEmail = EmailAddress.Text;
}
else
{
    request.DoEmailNotification = false;
}
```

Include the customers contact information in the interaction.

```
KeyValueCollection contactInfo = new KeyValueCollection();
contactInfo.Add("FirstName", FirstName.Text);
contactInfo.Add("LastName", LastName.Text);
contactInfo.Add("EmailAddress", EmailAddress.Text);

if (Media.Text.Equals("voice"))
{
    contactInfo.Add("PhoneNumber", PhoneNumber.Text);
}

request.BusinessAttributes = contactInfo;...
```

Display a single web callback interaction.

```
try
{
    WCBMessage response;
    response = WcbProtocol.Process(request, ISProtocol);

    if (response.Id == WCBResponseError.MessageId)
    {
        ShowErrorMessage(((WCBResponseError)response).ErrorMessage);
    }
    else if (response.Id == WCBResponseAck.MessageId)
    {
        EditWebCallback(((WCBResponseAck)response).InteractionId);
    }...
}
```

Retrieve a Single Web Callback Interaction

The following code retrieves a single web callback interaction using the `WCBRequestGetSingleInteraction` method:

```
protected void EditWebCallback(string id)
{
    WCBRequestGetSingleInteraction request = WCBRequestGetSingleInteraction.Create(id);

    try
    {
        WCBMessage response;
        response = WcbProtocol.Process(request, ISProtocol);

        if (response.Id == WCBResponseError.MessageId)
        {
            ShowErrorMessage(((WCBResponseError)response).ErrorMessage);
        }
        else if (response.Id == WCBResponseSingleInteraction.MessageId)
        {
            WebCallbackInteraction itx =
                ((WCBResponseSingleInteraction)response).WCBInteraction;

            SingleCallbackInfoPanel.Visible = true;
            TimeSpan timeShift = new TimeSpan(0, itx.TimeShift, 0);

            SWCB_ContactNumber_Label.Text = Server.HtmlEncode(itx.CustomerNumber);
            SWCB_Subject_Label.Text = Server.HtmlEncode(itx.Subject);
            SWCB_StartTime_Label.Text =
                itx.StartTime.Subtract(timeShift).ToLongTimeString();
            SWCB_EndTime_Label.Text = itx.EndTime.Subtract(timeShift).ToLongTimeString();
            SWCB_Media_Label.Text = itx.DesiredResponseType;
            SWCB_ID_Label.Text = id; ...
        }
    }
}
```

Retrieve a List of Web Callback Interactions

The following code retrieves a list of web callback interactions using the `WCBRequestGetMultipleInteractions` method:

```
protected void ShowMultipleWebCallbacks(string customerNumber)
{
    WCBRequestGetMultipleInteractions request =
        WCBRequestGetMultipleInteractions.Create(customerNumber);

    try
    {
        WCBMessage response;
        response = WcbProtocol.Process(request, ISProtocol);
        if (response.Id == WCBResponseError.MessageId)
        {
            ShowErrorMessage("No interactions found for Phone number (or IP) : " +
                customerNumber);
        }
        else if (response.Id == WCBResponseMultipleInteractions.MessageId)
        {
            ...
        }
    }
}
```

```

List<WebCallbackInteraction> interactions =
    ((WCBResponseMultipleInteractions)response).WCBInteractions;
if (interactions.Count == 0)
{
    ShowErrorMessage("No interactions found for Phone number (or IP) : " +
        customerNumber);
}
else
{
    DataTable dt = new DataTable();
    DataRow dr = null;

    dt.Columns.Add(new DataColumn("ID", typeof(string)));
    dt.Columns.Add(new DataColumn("CustomerNumber", typeof(string)));
    dt.Columns.Add(new DataColumn("Subject", typeof(string)));
    dt.Columns.Add(new DataColumn("Media", typeof(string)));
    dt.Columns.Add(new DataColumn("Start time", typeof(string)));
    dt.Columns.Add(new DataColumn("End time", typeof(string)));

    foreach (WebCallbackInteraction itx in interactions)
    {
        TimeSpan timeShift = new TimeSpan(0, itx.TimeShift, 0);
        dr = dt.NewRow();
        dr["ID"] = itx.InteractionId;
        dr["CustomerNumber"] = Server.HtmlEncode(itx.CustomerNumber);
        dr["Subject"] = Server.HtmlEncode(itx.Subject);
        dr["Media"] = itx.DesiredResponseType;
        dr["Start time"] = itx.StartTime.Subtract(timeShift).ToLongTimeString();
        dr["End time"] = itx.EndTime.Subtract(timeShift).ToLongTimeString();
        dt.Rows.Add(dr);
    }

    Session.Add("MultipleWebCallbackDataList", dt);
    MultipleWebCallbackDataList.DataSource =
        (DataTable)Session["MultipleWebCallbackDataList"];
    MultipleWebCallbackDataList.DataBind();

    MultipleCallbackCustomerNumberLabel.Text = customerNumber;
    MultipleWebCallbackInfoPanel.Visible = true;...
}

```

Cancel a Web Callback Interaction

The following code cancels a web callback interaction using the `WCBRequestCancel` method and displays the appropriate message:

```

protected void CancelWebCallback(string id)
{
    WCBRequestCancel request = WCBRequestCancel.Create(id);

    try

```

```

    {
        WCBMessage response;
        response = WcbProtocol.Process(request, ISProtocol);
        if (response.Id == WCBResponseError.MessageId)
        {
            ShowErrorMessage(((WCBResponseError)response).ErrorMessage);
        }
        else if (response.Id == WCBResponseAck.MessageId)
        {
            WebCallbackIsCancelledPanel.Visible = true;
        }
    }
    catch (Exception ex)
    {
        ShowErrorMessage(ex.Message);
        return;
    }
}

```

Reschedule a Web Callback Interaction

The following code reschedules a web callback interaction using the `WCBRequestReschedule` method:

```

protected void RescheduleWebCallback(string id, DateTime startTime, DateTime endTime)
{
    WCBRequestReschedule request = WCBRequestReschedule.Create(id);
    request.StartTime = startTime;
    request.EndTime = endTime;
    request.TimeShift = Convert.ToInt32(TimeShift.Value);

    try
    {
        WCBMessage response;
        response = WcbProtocol.Process(request, ISProtocol);

        if (response.Id == WCBResponseError.MessageId)
        {
            ShowErrorMessage(((WCBResponseError)response).ErrorMessage);
        }
        else if (response.Id == WCBResponseAck.MessageId)
        {
            EditWebCallback(id);
        }
    }
    catch (Exception ex)
    {
        ShowErrorMessage(ex.Message);
        return;
    }
}

```

Chat Sample

This section presents the sample's purpose, functionality overview, code implementation, and customization information.

Purpose

The Chat Sample demonstrates how to add a simple chat feature to any web form that supports .NET Active Server Pages functionality.

Functionality Overview

The following sections review the code used in implementing the different chat functions:

- “Declaring and Importing Packages” on [page 242](#)
- “Declaring Variables” on [page 242](#)
- “Retrieving Form Data” on [page 243](#)
- “Connecting to Chat Server” on [page 243](#)
- “Creating a Chat Session” on [page 257](#)
- “Handling Content” on [page 247](#)
- “Closing the Connection” on [page 251](#)

Files

The ...\\Chat directory contains the HTML Chat Sample. The sample consists of five files:

- ChatCommand.aspx and ChatCommand.aspx.cs—files that contain most of the chat logic.
- ChatFrameSet.htm—a frameset that holds the ChatCommand.aspx and ChatPanel.aspx files.
- ChatPanel.aspx and ChatPanel.aspx.cs—files that contain the code to create a chat panel with input boxes for entering the chat message. All data is sent to the parent form.

Code Explanation

The Chat Sample separates user interface and logic components. The subsections below explain the specific functions and the code for each of these components.

User Interface Implementation

The interface code demonstrates how to present form controls such as panels, input boxes, buttons, and so on. The user initially accesses these controls through the `ChatFrameset.htm` page, but the code resides primarily in the `ChatPanel.aspx` and, to a lesser extent, `ChatPanel.aspx.cs` files. The code is divided into these functions:

- “Drawing the Form” on [page 239](#)
- “Handling Content” on [page 240](#)

Drawing the Form

The `ChatPanel.aspx` file contains several JavaScript functions to handle connection state when the page is loaded or unloaded:

```
function window_onload ()
{
    bCommandFrameReady= false;
    disconnected();
}
```

Next, the `ChatPanel.aspx` file provides HTML that draws the page, gathers user information, provides buttons to call the functions that start and stop chat, and displays the chat transcript:

```
<form action="ChatPanel.aspx" id="chat_form" onsubmit="javascript:on_send(); return
false;">
  <table style="width:100%" cellpadding="2">
    <tr>
      <td class="samplesSampleTopToolbar">
        <table>
          <tr>
            <td style="width:435px"
class="samplesSampleNameTitleWhite">Simple Chat</td>
            <td style="width:91px" onclick="javascript:on_connect();"
class="sampleGreenButtonBig" onmouseover="javascript:MouseOver(this);"
onmouseout="javascript:MouseOut(this);">Start</td>
            <td style="width:91px" onclick="javascript:on_disconnect();"
class="sampleRedButtonBig" onmouseover="javascript:MouseOver(this);"
onmouseout="javascript:MouseOut(this);">Stop</td>
            <td style="width:auto" ></td>
          </tr>
        </table>
      </td>
    </tr>
    <tr>
      <td class="samplesSampleParagraph">Fill in your personal information in
fields below to proceed.</td>
    </tr>
```

```

    <tr>
      <td>
        <table style="width:617px" border="0">
          <tr>
            <td class="samplesSampleFieldTitle" ></td>
            <td></td>
            <td style="width:6px; vertical-align: middle;"></td>
          </tr>
          <tr>
            <td class="samplesSampleFieldTitle" >First name:</td>
            <td><input class="samplesFieldStyle" type="text" name="FirstName"
value=""/></td>
            <td style="width:6px; vertical-align: middle;"><span
class="samplesMandatoryMark">*</span></td>
          </tr>...

```

The `ChatPanel.aspx.cs` file contains a single class (`ChatSample_ChatPanel`) containing the single `Page_Load()` function:

```
protected void Page_Load(object sender, EventArgs e)
```

Handling Content

The `ChatPanel.aspx` file presents and gathers most of the information exchanged with the user. This section includes these subtopics:

- “Declarations and Header” on [page 240](#)
- “Indirect Functions for User Events” on [page 241](#)
- “Functions to Handle User Events” on [page 241](#)

Declarations and Header

The `ChatPanel.aspx` file begins by presenting an IIS directive, a DTD declaration, and an HTML header:

```

<%@ Page Language="C#" AutoEventWireup="true" CodeFile="ChatPanel.aspx.cs"
Inherits="ChatSample_ChatPanel" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
  <link rel="stylesheet" type="text/css" href="..\Resources/StyleSheet.css"/>
  <title>Genesys Multimedia 8.0.21 Samples. Chat.</title>
</head>

```

Indirect Functions for User Events

Next, the `ChatPanel.aspx` file contains this basic function:

- `window_onload()`—sets the logic when the window is first loaded into memory.

The chat panel resides in the frame named `itf`. You can invoke these functions from that frame:

- `on_connect()`—calls the `on_connect()` function in the page residing in the `itf` frame.
- `on_disconnect()`—calls the `on_disconnect()` function in the page residing in the `itf` frame.
- `on_send()`—calls the `on_send()` function in the page residing in the `itf` frame. Also prevents new requests from being sent to Chat Server until the response from the previous request has arrived.
- `message_onkeypress()`—sets the `bCommandFrameReady` variable to `false` and calls the `on_user_typing()` function on the page residing in the `itf` frame.

Functions to Handle User Events

The separate `ChatCommand.aspx` file contains five JavaScript functions to handle user events:

- `window_onload()`—calls either the `connected()` or `disconnected()` methods in the main form, depending on the value of the `itf_response` variable.
- `on_connect()`—sets the `cmd` variable to `connect`, sets the necessary data to connect to Chat Server, and calls the HTML form `submit()` function.
- `on_disconnect()`—sets the `cmd` variable to `disconnect` and calls the HTML form `submit()` function.
- `on_send()`—sets the `cmd` variable to `send`, sets the message to send with the value from the `strMessage` argument, and then calls the HTML form `submit()` function.
- `on_refresh()`—sets the `cmd` variable to `send` and calls the HTML form `submit()` function.
- `on_user_typing()`—calls the `parent.main.CommandFrameReady()` function. Sets the `cmd` variable to `user_typing`, the `msg2send` variable to `""`, and calls the HTML form `submit()` function.

Finally, the HTML code creates input boxes for users to fill in their personal information and their message:

```
<form method="post" action="ChatCommand.aspx" runat="server">
<asp:TextBox ID="cmd" AutoPostBack="false" Text="" runat="server" />
<asp:TextBox ID="chat_alias" AutoPostBack="false" Text="" runat="server" />
<asp:TextBox ID="first_name" AutoPostBack="false" Text="" runat="server" />
<asp:TextBox ID="last_name" AutoPostBack="false" Text="" runat="server" />
...
```

Logic Implementation

The ChatCommand.aspx.cs file contains the Chat Sample's main logic. The subsections below explain the code in this file.

Declaring and Importing Packages

The ChatCommand.aspx.cs file begins by declaring external packages:

```
using Genesyslab.Platform.Commons.Collections;
using Genesyslab.Platform.Commons.Protocols;
using Genesyslab.Platform.WebMedia.Protocols;
using Genesyslab.Platform.WebMedia.Protocols.FlexChat;
using Genesyslab.Platform.WebMedia.Protocols.FlexChat.Events;
using Genesyslab.Platform.WebMedia.Protocols.FlexChat.Requests;
using Genesyslab.WebApi.Core.ConfigServer;
using Genesyslab.WebApi.Core;
using Genesyslab.Platform.Configuration.Protocols.ConfServer;
using Genesyslab.Platform.Configuration.Protocols.Types;
```

Declaring Variables

Next, the ChatCommand.aspx.cs file's ChatSample_ChatCommand class declares internal variables. These variables represent data that the user enters in the form drawn by ChatCommand.aspx:

```
public partial class ChatSample_ChatCommand : System.Web.UI.Page
{
    string str_cmd                = "";
    string str_chat_alias        = "";
    string str_first_name        = "";
    string str_last_name         = "";
    string str_email_address     = "";
    string str_secure_key        = "";
    string str_user_id           = "";
    string str_session_id        = "";
    string str_timeZoneOffset    = "";
    string str_script_pos        = "";
    string str_msg2send          = "";
    string str_subject            = "";
    string str_itf_response      = "UNDEFINED";
    string str_itf_message       = "";
    string str_QueueKey          = "Chat inbound queue";
    FlexTranscript transcript    = null;
    bool clear_transcript        = false;
    string svcHost               = "";
    int svcPort                  = -1;
    bool bServiceAvailable       = false;
    SimpleSamplesConstants ssc    = new SimpleSamplesConstants(); ...
}
```

Retrieving Form Data

The `Page_Load()` function executes when the application displays pages or the user submits form data. It collects user data from the submitted form into the variables declared above:

```
protected void Page_Load(object sender, EventArgs e)
{
    str_cmd           = cmd.Text;
    str_chat_alias    = chat_alias.Text;
    str_first_name    = first_name.Text;
    str_last_name     = last_name.Text;
    str_email_address = email_address.Text;
    str_secure_key    = secure_key.Text;
    str_user_id       = user_id.Text;
    str_session_id    = session_id.Text;
    str_timeZoneOffset = timeZoneOffset.Text;
    str_script_pos     = script_pos.Text;
    str_msg2send      = msg2send.Text;
    str_subject       = subject.Text;
    IMessage imResponse = null; ...
}
```

Identifying the Chat Party

Within the subsequent try block, this branch ensures that the first- and last-name fields are filled, so that Chat Server will be able to identify this user:

```
if (str_first_name == "" || str_last_name == "")
{
    str_itf_response = "USERNAMEREQUIRED";
    str_itf_message = "Please enter first and last names.";
    ...
}
```

Enabling Custom Logging

This statement logs subsequent requests into a Genesys log file. It is an example of how to enable custom log messages:

```
LogMessage(LogLevel.Trace, "Starting new chat session.");
```

Connecting to Chat Server

Chat Server must maintain a live connection between users. This is unlike e-mail interactions, in which users fill out a form, submit the form, and thereby end the transaction.

If the load-balancing servlet cannot return an available Chat Server, the sample code informs the user:

```
try
{
```

```

        bServiceAvailable = false;
        ServiceInfo si = LoadBalancer.GetServiceInfo
            (CfgAppType.CFGChatServer, ssc.TenantName);
        svcHost = si.Host;
        svcPort = si.WebApiPort;
        str_chat_alias = si.Alias;
        bServiceAvailable = true;
    }
    catch (LoadBalancerException e1)
    {
        str_itf_response = "ERROR";
        str_itf_message = "Chat service is not available at this time.
            Please try it later.";
        LogMessage(LogLevel.Trace, str_itf_message);
    }

```

If the load balancer finds a Chat Server, the code returns an alias to that Chat Server and tries to connect and log the user in:

```

    if (bServiceAvailable == true)
    {
        Uri chatServerURI = new Uri("tcp://" + svcHost + ":" +
            svcPort.ToString());
        Endpoint chatEndPoint = new Endpoint(chatServerURI);
        FlexChatProtocol chat = new FlexChatProtocol(chatEndPoint);
        chat.Open();

        KeyValueCollection userdata = new KeyValueCollection();
        userdata.Add("FirstName", str_first_name);
        userdata.Add("LastName", str_last_name);
        if (str_email_address != null && str_email_address != "")
            userdata.Add("EmailAddress", str_email_address);

        string strNickName = str_first_name;
        if (str_last_name != null && str_last_name.Length > 0)
            strNickName = str_first_name + str_last_name.Substring(0, 1);
        RequestLogin rl = RequestLogin.Create(strNickName, 8, userdata);
        rl.TimezoneOffset = int.Parse(str_timeZoneOffset);
        LogMessage(LogLevel.Trace, "Sending RequestLogin to
            chat server.");
    }

```

Analyzing the Response

If Chat Server accepts the connection request, the code displays a welcome message:

```

    if (imResponse != null && imResponse.Name ==
        EventStatus.MessageName)
    {
        EventStatus status = imResponse as EventStatus;
    }

```

```

if (status.Id != 0 && status.SecureKey != "")
{
    LogMessage(LogLevel.Trace,
        "Logged to chat server.USERID=" + status.UserId);
    str_secure_key = status.SecureKey;
    str_user_id = status.UserId;
    str_itf_response = "CONNECTED";
    str_itf_message = "Welcome to Genesys chat!";
}

```

Joining and Transcribing the Chat Session

The next block attempts to join a chat session. If this request succeeds, the user joins the session, and the application begins collecting the chat transcript for later processing. (This deferred processing differs from the linear processing in the corresponding Java “Chat Sample” on [page 121](#).)

```

LogMessage(LogLevel.Trace, "Trying to Join to session for user with
    USERID= " + str_user_id);
RequestJoin rj = RequestJoin.Create(str_user_id, str_secure_key, "", ssc.TenantName +
    ":" +
    str_QueueKey, str_subject);
imResponse = chat.Request(rj, new TimeSpan(0, 0, 30));
EventStatus es = imResponse as EventStatus;

if (es != null && es.RequestResult == RequestResult.Success)
{
    LogMessage(LogLevel.Trace, "Joined to session for user with USERID="
        + str_user_id + " and SESSIONID=" + es.FlexTranscript.SessionId);
    clear_transcript = true;
    transcript = es.FlexTranscript;
    str_script_pos = es.FlexTranscript.LastPosition.ToString();
    str_itf_response = "CONNECTED";
}

```

Handling Connection and Session Errors

If Chat Server fails to accommodate the connection or the join-session requests, the following code relays and logs Chat Server’s error messages. The first (inner) branch handles chat-session errors, and the second (outer) branch connection handles errors:

```

else
{
    str_itf_response = "DISCONNECTED";

    if (es.Description != null)
    {
        str_itf_message = es.Description.Text;
        LogMessage(LogLevel.Trace, "Can't join to

```

```

        session for user with USERID=" + str_user_id + ". Reason
        " + str_itf_message);
    }
    else
    {
        str_itf_message = "Could not create chat session.";
        LogMessage(LogLevel.Trace, "Can't join to
        session for user with USERID=" + str_user_id);
    }
}
else
{
    str_itf_response = "DISCONNECTED";

    if (status.Description != null)
    {
        str_itf_message = status.Description.Text;
        LogMessage(LogLevel.Trace, "Can't connect to chat server.
        Reason " + str_itf_message);
    }
    else
    {
        str_itf_message = "Not connected to chat server, unexpected error.";
        LogMessage(LogLevel.Trace, "Can't connect to chat server.");
    }
}
}
else
{
    str_itf_response = "ERROR";
    str_itf_message = "chat.lasterror()";
}
if (chat != null)
{
    chat.Close();
    chat = null;
}
}
}
catch (Exception cex)
{
    // failed to connect to chat server
    str_itf_response = "NOSERVER";
    str_itf_message = "Failed to establish Chat ('" + cex.ToString() + "')";
    LogMessage(LogLevel.Exception, cex.ToString());
}

```

Handling Content

This section covers these topics:

- “Handling User Requests” on [page 247](#)
- “Processing Server Events” on [page 249](#)
- “Masking Data” on [page 249](#)

Handling User Requests

The `cmd` variable reflects the action or button the user clicked. If the user clicked the Connect button, then the code attempts to get a handle to the load balancer:

```
if (str_chat_alias != "" && str_cmd != "connect")
{
    try
    {
        bServiceAvailable = false;
        ServiceInfo si = LoadBalancer.GetServiceInfo(str_chat_alias);
        svcHost = si.Host;
        svcPort = si.WebApiPort;
        bServiceAvailable = true;
    }
    catch (LoadBalancerException e1)...
```

Other possible requests are to disconnect from the server or to send a chat message. The application provides code to submit, and to respond to exceptions from, each type of request. Here is the code for a disconnect request:

```
if (str_cmd == "disconnect")
{
    LogMessage(LogLevel.Trace, "Sending
    Request Logout for user with USERID=" + str_user_id);
    RequestLogout rlo = RequestLogout.Create(str_user_id,
        str_secure_key, 0);
    imResponse = chat.Request(rlo, new TimeSpan (0, 0, 30));
    EventStatus es = imResponse as EventStatus;

    str_itf_response = "DISCONNECTED";
    str_itf_message = "Chat was finished";
}...
```

Here is the code to handle a `user_typing` request:

```
else if (str_cmd == "user_typing")
{
    EventStatus es = null;
    try
    {
        LogMessage(LogLevel.Trace,
            "Sending RequestRefresh with
            \"user typing notification\" for user with USERID="
            + str_user_id);
```

```

RequestRefresh rr =
    RequestRefresh.Create(str_user_id, str_secure_key,
        int.Parse(str_script_pos) + 1, MessageText.Create
            ("text", TreatAs.SYSTEM, "user is typing"));
imResponse = chat.Request(rr, new TimeSpan(0, 0, 30));
es = imResponse as EventStatus;
transcript = es.FlexTranscript;
str_script_pos = es.FlexTranscript.LastPosition.ToString();
str_itf_response = "TRANSCRIPT";
}
catch (Exception ex)
{
    str_itf_response = "SENDFAILED";
    if (es != null && es.Description.Text != null)
        str_itf_message = es.Description.Text;
    else
        str_itf_message = "Can't get chat transcript
            from incoming packet. " + ex.ToString();
    LogMessage(LogLevel.Exception, str_itf_message);
}
}

```

Here is the code to handle a send request:

```

else if (str_cmd == "send")
{
    EventStatus es = null;
    try
    {
        LogMessage(LogLevel.Trace, "Sending
            RequestRefresh for user with USERID=" + str_user_id);
        RequestRefresh rr = RequestRefresh.Create(str_user_id,
            str_secure_key, int.Parse(str_script_pos) + 1,
            MessageText.Create("text", str_msg2send == "" ? null :
                (str_msg2send)));

        imResponse = chat.Request(rr, new TimeSpan(0, 0, 30));
        es = imResponse as EventStatus;

        transcript = es.FlexTranscript;
        str_script_pos =(es.FlexTranscript.LastPosition.ToString());
        str_itf_response = "TRANSCRIPT";
    }
    catch (Exception ex)
    {
        str_itf_response = "SENDFAILED";
        if (es != null && es.Description.Text != null)
            str_itf_message = es.Description.Text;
        else
            str_itf_message = "Can't get chat transcript from

```

```

        incoming packet. " + ex.ToString();
        LogMessage(LogLevel.Exception, str_itf_message);
    }...

```

Processing Server Events When the sample receives a server event, the code uses `EventType` values to check the current status of the chat packet. In the following code snippet, the code checks for connection, abandonment, and message flags:

```

if (chat_event.EventType == EventType.Connect)
{
    if (chat_event.UserType != UserType.External)
    {
        text2append = text2append + "New party ('" + chat_event.UserNickname + "')
            has joined the session";

        if (chat_event.Text != "")
            text2append = text2append + ": " + chat_event.Text;
    }
}
else if (chat_event.EventType == EventType.Message)
{
    if (chat_event.Text != null)
        text2append = text2append + chat_event.UserNickname + ": " + chat_event.Text;
    else
        text2append = text2append + chat_event.UserNickname + ":";
}
else if (chat_event.EventType == EventType.Abandon)
{
    text2append = text2append + "Party ('" + chat_event.UserNickname+"')
        has left the session.";}

```

Masking Data The `MaskSymbols()` function filters out any characters that are potentially dangerous, or that cannot be processed by JavaScript, in the client's browser:

```

public string MaskSymbols (string strIn)
{
    string strOut = "";
    int i;
    string ch = "";

    if (strIn != null)
    {
        for (i = 0; i < strIn.Length; i++)
        {
            ch = strIn.Substring (i, 1);
            if (ch == "\r")
                strOut += "\\r";
            else if (ch == "\n")
                strOut += "\\n";
            else if (ch == "\t")
                strOut += "\\t";

```

```

        else if (ch == "\\")
            strOut += "\\\"";
        else if (ch == "\\")
            strOut += "\\\"";
        else if (ch == "<")
        {
            if (i != (strIn.Length - 1))
                strOut += "\" + \"<\" + \"\"";
            else
                strOut += "\" + \"<\"";
        }
        else if (ch == ">")
        {
            if (i != (strIn.Length - 1))
                strOut += "\" + \">\" + \"\"";
            else
                strOut += "\" + \">\"";
        }
        else
            strOut += ch;
    }
}
return strOut;
}

```

The `mask_html()` function similarly traps and converts HTML-safe characters:

```

public string mask_html (string strIn)
{
    string strOut = "";
    int i;
    string ch;
    if (strIn != null)
    {
        for (i=0; i < strIn.Length; i++)
        {
            ch = strIn.Substring (i, 1);
            if (ch == "<")
                strOut += "&lt;";
            else if (ch == ">")
                strOut += "&gt;";
            else if (ch == "\r")
                strOut += " ";
            else if (ch == "\n")
                strOut += " ";
            else if (ch == "\"")
                strOut += "&quot;";
            else if (ch == "&")
                strOut += "&amp;";
            else
                strOut += ch;
        }
    }
}

```

```
        }  
    }  
    return strOut;  
}  
}
```

Closing the Connection

When the processing is done and the server is ready to send back its response, the connection to the Chat Server must end and the service dispatcher must be reset to `null`. Otherwise, you will have orphan connections to the server that do not relinquish resources, principally network resources (like sockets) and memory. This connection-release code occurs in the `else` block of “Creating a Chat Session” on [page 257](#):

```
if( chat != null )  
{  
    chat.close();  
    chat = null;  
}
```

In addition, before ending a chat by logging out of Chat Server, your client application’s code must ensure that the application waits to receive a reply from Chat Server to the browser’s `Connect` request. Otherwise, the logged-out client will leave behind a pending interaction that Genesys Desktop will be unable to delete.

Chat with AJAX Sample

This section presents the sample’s purpose, functionality overview, code implementation, and customization information.

Purpose

The Chat with AJAX Sample demonstrates how to add a simple chat feature to any web form using AJAX technology.

Functionality Overview

The following sections review the code used in implementing the different chat functions:

- “Declaring and Importing Packages” on [page 254](#)
- “Declaring Variables” on [page 255](#)
- “Retrieving Form Data” on [page 255](#)
- “Connecting to Chat Server” on [page 256](#)

- “Creating a Chat Session” on [page 257](#)
- “Handling Content” on [page 259](#)
- “Closing the Connection” on [page 261](#)

Files

The `...\ChatAjax` directory contains the Chat with AJAX Sample. The sample consists of four files:

- `ChatCommand.aspx` and `ChatCommand.aspx.cs`—files that contain most of the chat logic.
- `ChatPanel.aspx` and `ChatPanel.aspx.cs`—files that contain the code to create a chat panel with input boxes for entering the chat message. All data is sent to the hidden object that passes this data to the server.

Code Explanation

The Chat with AJAX Sample separates user interface and logic components. The subsections below explain the specific functions and the code for each of these components. The code used to create this example is similar to the basic “Chat Sample” on [page 238](#). The most notable difference being that this sample returns a JSON object instead of JavaScript code.

User Interface Implementation

The interface code demonstrates how to present form controls such as panels, input boxes, and buttons. The code resides in the `ChatPanel.aspx` and, to a lesser extent, `ChatPanel.aspx.cs` files. The code is divided into these functions:

- “Drawing the Form” on [page 252](#)
- “Handling Content” on [page 247](#)

Drawing the Form

The `ChatPanel.aspx` file contains several JavaScript functions to handle connection state when the page is loaded or unloaded:

```
function window_onload ()
{
    disconnected();
}
```

Next, the `ChatPanel.aspx` file provides HTML that draws the page, gathers user information, provides buttons to call the functions that start and stop chat, and displays the chat transcript:

```

...<tr>
    <td class="samplesSampleTopToolbar">
        <table >
            <tr >
                <td style="width:435px" class="samplesSampleNameTitleWhite">Chat
AJAX</td>
                <td style="width:91px" onclick="javascript:on_connect();"
class="sampleGreenButtonBig" onmouseover="javascript:MouseOver(this);"
onmouseout="javascript:MouseOut(this);">Start</td>
                <td style="width:91px" onclick="javascript:on_disconnect();"
class="sampleRedButtonBig" onmouseover="javascript:MouseOver(this);"
onmouseout="javascript:MouseOut(this);">Stop</td>
                <td style="width:auto" ></td>
            </tr>
        </table>
    </td>
</tr>...

...<tr>
    <td class="samplesSampleFieldTitle" >First name:</td>
    <td><input class="samplesFieldStyle" type="text" name="FirstName"
value=""/></td>
    <td style="width:6px; vertical-align: middle;"><span
class="samplesMandatoryMark">*</span></td>
</tr>

    <tr>
        <td class="samplesSampleFieldTitle" >Last name:</td>
        <td><input class="samplesFieldStyle" type="text" name="LastName"/
value=""/></td>
        <td style="width:6px; vertical-align: middle;"><span
class="samplesMandatoryMark">*</span></td>

</tr>...

```

The `ChatPanel.aspx.cs` file contains a single class (`ChatSample_ChatPanel`) containing the single `Page_Load()` function:

```
protected void Page_Load(object sender, EventArgs e)
```

Handling Content

The `ChatPanel.aspx` file presents and gathers most of the information exchanged with the user. This section includes these subtopics:

- “Declarations and Header” on [page 253](#)
- “Functions to Handle User Events” on [page 254](#)

Declarations and Header The `ChatPanel.aspx` file begins by presenting an IIS directive, a DTD declaration, and an HTML header:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="ChatPanel.aspx.cs"
    Inherits="ChatSample_ChatPanel" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
    <link rel="stylesheet" type="text/css" href="../../Resources/StyleSheet.css"/>
    <title>Genesys Multimedia 8.0.21 Samples. Chat with AJAX.</title>
</head>
```

Functions to Handle User Events

The ChatPanel.aspx file contains these basic function to handle events:

- window_onload()—calls the disconnected() method.
- on_connect()—sets the str_cmd variable to connect, initializes the date, time, first name, last name, email address, and subject variables, and calls the httpRequest() function.
- on_disconnect()—sets the str_cmd variable to disconnect and calls the httpRequest() function.
- on_send()—sets the str_cmd variable to send, sets the str_msg2send variable with the value from the form's message field, and then calls the httpRequest() function.
- on_refresh()—sets the str_cmd variable to send, sets the str_msg2send variable to blank, and calls the httpRequest() function.

Logic Implementation

The ChatCommand.aspx.cs file contains the Chat with AJAX Sample's main logic. The subsections below explain the code in this file.

Declaring and Importing Packages

The ChatCommand.aspx.cs file begins by declaring external packages:

```
using System;
using System.Data;
using System.Configuration;
using System.Collections;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using Genesyslab.Platform.Commons.Collections;
using Genesyslab.Platform.Commons.Protocols;
using Genesyslab.Platform.WebMedia.Protocols;
using Genesyslab.Platform.WebMedia.Protocols.FlexChat;
using Genesyslab.Platform.WebMedia.Protocols.FlexChat.Events;
```

```

using Genesyslab.Platform.WebMedia.Protocols.FlexChat.Requests;
using Genesyslab.WebApi.Core.ConfigServer;
using Genesyslab.WebApi.Core;
using Genesyslab.Platform.Configuration.Protocols.ConfServer;
using System.Collections.Specialized;

```

Declaring Variables

Next, the `ChatCommand.aspx.cs` file's `ChatSample_ChatCommand` class declares internal variables. These variables represent data that the user enters in the form drawn by `ChatCommand.aspx`:

```

public partial class ChatSample_ChatCommand : System.Web.UI.Page
{
    string str_cmd                = "";
    string str_chat_alias         = "";
    string str_first_name        = "";
    string str_last_name         = "";
    string str_email_address     = "";
    string str_secure_key        = "";
    string str_user_id           = "";
    string str_session_id        = "";
    string str_timeZoneOffset    = "";
    string str_script_pos        = "";
    string str_msg2send          = "";
    string str_subject           = "";
    string str_itf_response       = "UNDEFINED";
    string str_itf_message       = "";
    string str_QueueKey          = "Chat inbound queue";
    FlexTranscript transcript    = null;
    bool clear_transcript        = false;
    string svcHost               = "";
    int svcPort                  = -1;
    bool bServiceAvailable       = false;
    SimpleSamplesConstants ssc    = new SimpleSamplesConstants(); ...
}

```

Retrieving Form Data

The `Page_Load()` function executes when the application displays pages or the user submits form data. It collects user data from the submitted form into the variables declared above:

```

protected void Page_Load(object sender, EventArgs e)
{
    NameValueCollection qs = Request.Form;
    if (qs.Count == 0)
        qs = Request.QueryString;
    str_cmd                = qs.Get("cmd");
    str_chat_alias         = qs.Get("chat_alias");
    str_first_name         = qs.Get("first_name");
}

```

```

        str_last_name          = qs.Get("last_name");
        str_email_address     = qs.Get("email_address");
        str_secure_key        = qs.Get("secure_key");
        str_user_id           = qs.Get("user_id");
        str_session_id        = qs.Get("session_id");
        str_timeZoneOffset    = qs.Get("timeZoneOffset");
        str_script_pos        = qs.Get("script_pos");
        str_msg2send           = qs.Get("msg2send");
        str_subject           = qs.Get("subject");
        IMessage imResponse   = null; ...

```

Identifying the Chat Party

Within the subsequent try block, this branch ensures that the first- and last-name fields are filled, so that Chat Server will be able to identify this user:

```

if (str_first_name == "" || str_last_name == "")
{
    str_itf_response = "USERNAMEREQUIRED";
    str_itf_message = "Please enter first and last names."; ...
}

```

Connecting to Chat Server

Chat Server must maintain a live connection between users. This is unlike e-mail interactions, in which users fill out a form, submit the form, and thereby end the transaction.

If the load-balancing servlet cannot return an available Chat Server, the sample code informs the user:

```

try
{
    bServiceAvailable = false;
    ServiceInfo si = LoadBalancer.GetServiceInfo
        (CfgAppType.CFGChatServer, ssc.TenantName);
    svcHost = si.Host;
    svcPort = si.WebApiPort;
    str_chat_alias = si.Alias;
    bServiceAvailable = true;
}
catch (LoadBalancerException e1)
{
    str_itf_response = "ERROR";
    str_itf_message = "Chat service is not available at this time.
        Please try it later.";
}

```

If the load balancer finds a Chat Server, the code returns an alias to that Chat Server and tries to connect and log the user in:

```

if (bServiceAvailable == true)
{
    Uri chatServerURI = new Uri("tcp://" + svcHost + ":" +
        svcPort.ToString());
    Endpoint chatEndPoint = new Endpoint(chatServerURI);
}

```

```

FlexChatProtocol chat = new FlexChatProtocol(chatEndPoint);
chat.Open();

KeyValueCollection userdata = new KeyValueCollection();
userdata.Add("FirstName", str_first_name);
userdata.Add("LastName", str_last_name);
if (str_email_address != null && str_email_address != "")
    userdata.Add("EmailAddress", str_email_address);

string strNickName = str_first_name;
if (str_last_name != null && str_last_name.Length > 0)
    strNickName = str_first_name + str_last_name.Substring(0, 1);
RequestLogin rl = RequestLogin.Create(strNickName, 8, userdata);
rl.TimezoneOffset = int.Parse(str_timeZoneOffset);
imResponse = chat.Request(rl, new TimeSpan(0, 0, 30));

```

Analyzing the Response

If Chat Server accepts the connection request, the code displays a welcome message:

```

if (imResponse != null && imResponse.Name ==
    EventStatus.MessageName)
{
    EventStatus status = imResponse as EventStatus;

    if (status.Id != 0 && status.SecureKey != "")
    {
        str_secure_key = status.SecureKey;
        str_user_id = status.UserId;
        str_itf_response = "CONNECTED";
        str_itf_message = "Welcome to Genesys chat!";...
    }
}

```

Creating a Chat Session

Joining and Transcribing the Chat Session

The next block attempts to join a chat session. If this request succeeds, the user joins the session, and the application begins collecting the chat transcript for later processing. (This deferred processing differs from the linear processing in the corresponding Java “Chat Sample” on [page 121](#).)

```

RequestJoin rj = RequestJoin.Create
    (str_user_id, str_secure_key, "",
     ssc.TenantName + ":" + str_QueueKey, str_subject);
imResponse = chat.Request(rj, new TimeSpan(0, 0, 30));
EventStatus es = imResponse as EventStatus;

if (es != null && es.RequestResult == RequestResult.Success)
{
    clear_transcript = true;
    transcript = es.FlexTranscript;
    str_script_pos = es.FlexTranscript.LastPosition.ToString();
    str_itf_response = "CONNECTED";
}

```

**Handling
Connection and
Session Errors**

If Chat Server fails to accommodate the connection or the join-session requests, the following code relays and logs Chat Server's error messages. The first (inner) branch handles chat-session errors, and the second (outer) branch connection handles errors:

```

else
{
    str_itf_response = "DISCONNECTED";

    if (es.Description != null)
    {
        str_itf_message = es.Description.Text;
    }
    else
    {
        str_itf_message = "Could not create chat session.";
    }
}
else
{
    str_itf_response = "DISCONNECTED";

    if (status.Description != null)
    {
        str_itf_message = status.Description.Text;
    }
    else
    {
        str_itf_message = "Not connected to chat server, unexpected
        error."; ...
    }
}
else
{
    str_itf_response = "ERROR";
    str_itf_message = "chat.lasterror()";
}
if (chat != null)
{
    chat.Close();
    chat = null;
}
catch (Exception cex)
{
    // failed to connect to chat server
    str_itf_response = "NOSERVER";
    str_itf_message = "Failed to establish Chat
    ('" + cex.ToString() + "')";
}

```

Handling Content

This section covers these topics:

- “Handling User Requests” on [page 259](#)
- “Processing Server Events” on [page 260](#)
- “Masking Data” on [page 261](#)

Handling User Requests

The `cmd` variable reflects the action or button the user clicked. If the user clicked the Connect button, then the code attempts to get a handle to the load balancer:

```
if (str_chat_alias != "" && str_cmd != "connect")
{
    try
    {
        bServiceAvailable = false;
        ServiceInfo si = LoadBalancer.GetServiceInfo(str_chat_alias);
        svcHost = si.Host;
        svcPort = si.WebApiPort;
        bServiceAvailable = true;
    }
    catch (LoadBalancerException e1)...
```

Other possible requests are to disconnect from the server or to send a chat message. This sample does not support `user_typing` notification, but it can be easily implemented. The application provides code to submit, and to respond to exceptions from, each type of request. Here is the code for a disconnect request:

```
if (str_cmd == "disconnect")
{
    RequestLogout rlo = RequestLogout.Create(str_user_id,
        str_secure_key, 0);
    imResponse = chat.Request(rlo, new TimeSpan (0, 0, 30));
    EventStatus es = imResponse as EventStatus;

    str_itf_response = "DISCONNECTED";
    str_itf_message = "Chat was finished";
}...
```

Here is the code to handle a send request:

```
else if (str_cmd == "send")
{
    EventStatus es = null;
    try
    {
```

```

RequestRefresh rr = RequestRefresh.Create(str_user_id,
    str_secure_key, int.Parse(str_script_pos) + 1,
    MessageText.Create("text", str_msg2send == "" ? null :
        str_msg2send));

imResponse = chat.Request(rr, new TimeSpan(0, 0, 30));
es = imResponse as EventStatus;

transcript = es.FlexTranscript;
str_script_pos =
    es.FlexTranscript.LastPosition.ToString();
str_itf_response = "TRANSCRIPT";
}
catch (Exception ex)
{
    str_itf_response = "SENDFAILED";
    if (es != null && es.Description.Text != null)
        str_itf_message = es.Description.Text;
    else
        str_itf_message = "Can't get chat transcript from
            incoming packet. " + ex.ToString();
}...

```

Processing Server Events

When the sample receives a server event, the code uses `EventType` values to check the current status of the chat packet. In the following code snippet, the code checks for connection, abandonment, and message flags:

```

if (chat_event.EventType == EventType.Connect)
{
    if (chat_event.UserType != UserType.External)
    {
        strText2Add = "\"New party ('" + MaskSymbols
            (chat_event.UserNickname) + "') has joined the session\"";
    }
}
else if (chat_event.EventType == EventType.Message)
{
    if (chat_event.Text != null)
        strText2Add = "\"\" + chat_event.UserNickname + ": \" +
            MaskSymbols(chat_event.Text) + "\"";
    else
        strText2Add = "\"\" + MaskSymbols(chat_event.UserNickname)
            + ":\"";
}
else if (chat_event.EventType == EventType.Abandon)
{
    strText2Add = "\"Party ('" + MaskSymbols
        (chat_event.UserNickname) + "') has left the session.\"";
}
if (strText2Add != null)
{

```

```

        strOut += strText2Add;
        if (iCount != transcript.EventInfoList.Count - 1)
            strOut += ",\r\n";
    }

```

Masking Data The `MaskSymbols()` function filters out any characters that are potentially dangerous, or that cannot be processed by JavaScript, in the client's browser:

```

public string MaskSymbols (string strIn)
{
    string strOut = "";
    int i;
    string ch = "";

    if (strIn != null)
    {
        for (i = 0; i < strIn.Length; i++)
        {
            ch = strIn.Substring (i, 1);
            if(ch == "\r") strOut += "\\r";
            else if (ch == "\n") strOut += "\\n";
            else if (ch == "\t") strOut += "\\t";
            else if (ch == "\"") strOut += "\\\"";
            else if (ch == "\\") strOut += "\\\"";
            else if (ch == "<")
            {
                if (i != (strIn.Length - 1))
                    strOut += "\" + \"<\" + \"\"";
                else
                    strOut += "\" + \"<\"";
            }
            else if (ch == ">")
            {
                if (i != (strIn.Length - 1))
                    strOut += "\" + \">\" + \"\"";
                else
                    strOut += "\" + \">\"";
            }
            else
                strOut += ch;
        }
    }
    return strOut;
}

```

Closing the Connection

When the processing is done and the server is ready to send back its response, the connection to the Chat Server must end and the service dispatcher must be reset to `null`. Otherwise, you will have orphan connections to the server that do not relinquish resources, principally network resources (like sockets) and

memory. This connection-release code occurs in the `else` block of “Creating a Chat Session” on [page 257](#):

```
if( chat != null )
{
    chat.close();
    chat = null;
}
```

In addition, before ending a chat by logging out of Chat Server, your client application’s code must ensure that the application waits to receive a reply from Chat Server to the browser’s `Connect` request. Otherwise, the logged-out client will leave behind a pending interaction that Genesys Desktop will be unable to delete.

Survey Sample

This section presents the purpose, functionality overview, and code implementation of the Survey Sample. This survey sample is included in the Advanced Chat Sample. Some references to the Advanced Chat Sample will be made to clarify the code that enables the survey to be displayed.

Purpose

The Survey Sample demonstrates how to include a survey in a chat window.

Functionality Overview

The following sections review the code that is used in the implementation of the different survey functions:

- “Displaying the Survey” on [page 263](#)
- “Gathering the Survey Results” on [page 264](#)
- “Submitting the Survey Results” on [page 266](#)

For information about code that is common to all of the samples—such as retrieving parameters, setting the content type, character encoding, loading libraries, and importing files—see “Common Functions” on [page 229](#).

Files

The `.../Survey` directory contains the Survey Sample. The sample consists of two files, `Survey.aspx` and `Survey.aspx.cs`.

Code Explanation

As in the other .NET samples, the .aspx file handles most of the page presentation to the user, while the .aspx.cs file contains most of the logic.

Displaying the Survey

The Advance Chat Sample displays a checkbox associated with survey on the chat window. When the box is checked, the survey will be presented to the customer once the chat session has been completed. The following code appears in the AdvancedChat/ChatPanel.aspx file to display the survey checkbox:

```
...Take survey after chat <input type="checkbox" id="SurveyAfterChat"
name="SurveyAfterChat" checked="checked" />...
```

The file also contains a user event handler and two functions associated with the survey functionality:

The event handler, TranscriptAfterChat_onclick(), determines if the checkbox is selected:

```
function TranscriptAfterChat_onclick()
{
    if (bConnected == true)
    {
        bCommandFrameReady = false;
        parent.itf.on_send_transcript(document.forms[0].SurveyAfterChat.checked);
    }
}
```

The GetSurveyUrl() function returns the survey's URL:

```
function GetSurveyUrl()
{
    var theURL = unescape(location.href);
    var iPos = theURL.indexOf("SimpleSamples8021", 0);
    if (iPos != -1)
    {
        theURL=theURL.substring(0, iPos);
        theURL = theURL + "SimpleSamples8021/" +
        "Survey/Survey.aspx?ParentID="+strSessionID;
    }
    else
    {
        theURL =
        "http://localhost/SimpleSamples8021/Survey/Survey.aspx?ParentID="+strSessionID;
    }
    return theURL;
}
```

The function, `disconnected()` calls `GetSurveyUrl()` and opens the survey in a new window. It also determines if a pop-blocker is being used and sends a message to suggest that the customer disable the blocker:

```
function disconnected()
{
    bConnected = false;
    document.forms[0].send.disabled = true;
    if (document.forms[0].SurveyAfterChat.checked && SessionID != "")
    {
        var strMessage = GetSurveyUrl(); //"../Survey/Survey.aspx?ParentID="+SessionID;
        var bNeedToShowMessage = false;
        var winNewWindow = window.open (strMessage, "survey", "", false);

        if (winNewWindow == null)
            bNeedToShowMessage = true;
        else
        {
            if (window.opera)
            {
                if (!winNewWindow.opera)
                    bNeedToShowMessage = true;
            }
        }

        if (bNeedToShowMessage)
        {
            window.frames.ChatTranscript.AddMessage ("System: pop-up blocker detected. ",
                AgentNickNameColor, 1, 1);
            window.frames.ChatTranscript.AddMessage ("Please navigate to the next link to
                take our survey: " + strMessage, AgentMessageColor, 0, 0);
        }
    }
}
```

Gathering the Survey Results

The `Survey.aspx` and `Survey.aspx.cs` files contain the code to create the survey form, and the event handlers and functions needed to gather the customer's selections and submit the results.

The following code snippet allows you to change the values of the radio buttons on the form:

```
<form action="Survey.aspx" id="survey_form" onsubmit="javascript:on_send(); return
    false;" runat="server" method="post" name="survey_form">
    <table style="width:100%" cellpadding="0" cellspacing="0">
        <tr>
            <td class="samplesSampleNameTitle">Survey. Please complete this short
            survey to rate your experience</td>
```

```

        </tr>

        <tr>
            <td>
                <table border="0">
                    <tr>
                        <td class="samplesSampleFieldTitleFontOnly" style="width:300px;
vertical-align: middle;color:#15428B;background-color:#DFE8F6;" >How would you rate
your overall experience:</td>
                        <td class="samplesSampleFieldTitleFontOnly"
style="color:#15428B;background-color:#DFE8F6;">
                            <asp:RadioButtonList
class="samplesSampleFieldTitleFontOnly" RepeatLayout="Table"
RepeatDirection="horizontal" TextAlign="right"
style="vertical-align:middle;color:#15428B;background-color:#DFE8F6;"
ID="RadioBtnList_Experience" runat="server">
                                <asp:ListItem Selected="True">Very
good</asp:ListItem>
                                <asp:ListItem>Good</asp:ListItem>
                                <asp:ListItem>Average</asp:ListItem>
                                <asp:ListItem>Poor</asp:ListItem>
                                <asp:ListItem>Very poor</asp:ListItem>
                            </asp:RadioButtonList>
                        </td>
                        <td style="width:6px; vertical-align: middle;"><font
class="samplesMandatoryMark">*</font></td>
                    </tr>
                </table>
            </td>
            <td>
                <table border="0">
                    <tr>
                        <td class="samplesSampleFieldTitleFontOnly" style="width:300px;
vertical-align: middle;">How would you rate the resolution we provided:</td>
                        <td class="samplesSampleFieldTitleFontOnly" >
                            <asp:RadioButtonList
class="samplesSampleFieldTitleFontOnly" RepeatDirection="horizontal"
TextAlign="right" style=" vertical-align: middle;" ID="RadioBtnList_Resolution"
runat="server">
                                <asp:ListItem Selected="True">Very
good</asp:ListItem>
                                <asp:ListItem>Good</asp:ListItem>
                                <asp:ListItem>Average</asp:ListItem>
                                <asp:ListItem>Poor</asp:ListItem>
                                <asp:ListItem>Very poor</asp:ListItem>
                            </asp:RadioButtonList>
                        </td>...
                    </tr>
                </table>
            </td>
        </tr>
    </table>

```

The `window_onload()` event handler is called every time that the page is loaded. This function clears any selected radio buttons on the form when it is initially opened:

```

function window_onload ()
{
    document.forms[0].ParentID.value = "<%=strParentID%>";
    document.forms[0].ContactID.value = "<%=strContactID%>";
    document.forms[0].TenantID.value = "<%=strTenantID%>";

    applyCoolStyle("RadioBtnList_Experience_0");
    applyCoolStyle("RadioBtnList_Experience_1");
    applyCoolStyle("RadioBtnList_Experience_2");
    applyCoolStyle("RadioBtnList_Experience_3");
    applyCoolStyle("RadioBtnList_Experience_4");

    applyCoolStyle("RadioBtnList_Resolution_0");
    applyCoolStyle("RadioBtnList_Resolution_1");
    applyCoolStyle("RadioBtnList_Resolution_2");
    applyCoolStyle("RadioBtnList_Resolution_3");
    applyCoolStyle("RadioBtnList_Resolution_4");

    applyCoolStyle("RadioBtnList_Recommend_0");
    applyCoolStyle("RadioBtnList_Recommend_1");

    Custom.init();
}

```

Submitting the Survey Results

This section of code creates an instance of Load Balancer and an instance of Interaction Server by using the host and port IDs of Interaction Server. If there are any errors or exceptions, the code handles them:

```

try
{
    ServiceInfo si =
LoadBalancer.GetServiceInfo(CfgAppType.CFGInteractionServer, ssc.TenantName);
    strInteractionServerHost = si.Host;
    iInteractionServerPort = si.Port;
}
catch (LoadBalancerException e1)
{
    strMessage = "Interaction server is not available at this time.
Please try it later.";
}
if (strInteractionServerHost != "" && iInteractionServerPort != -1)
{
    if (Connect(strInteractionServerHost, iInteractionServerPort))...

```

Once connected, retrieve the tenantID:

```

if (strTenantID != null && strTenantID != "")
    reqSubmit.TenantId = int.Parse(strTenantID);
else
    reqSubmit.TenantId =
int.Parse(LoadBalancer.getTenantId(ssc.TenantName));

```

Create the reqSubmit object:

```

...try
{
RequestSubmit reqSubmit = RequestSubmit.Create();

```

A collection of key-value pairs containing the survey questions and answers. For example, the key for a question is `survey_question_FIELDNAME`. The answer to this question is stored in `survey_answer_FIELDNAME` key. The order number of the question is stored in `survey_question_number_FIELDNAME` key.

The prefixes `survey_question_`, `survey_answer_` and `survey_question_number_` are hardcoded and allow us to separate survey related data from other information in the attached data. When processing the user data you may match the original question with a correct answer by the common postfix `FIELDNAME`.

```

KeyValueCollection userData = new KeyValueCollection();
userData.Set(ssc.SurveyQuestionPrefix + "Experience", "How would you rate your overall
    experience");
userData.Set(ssc.SurveyAnswerPrefix + "Experience", strExperienceAnswer);
userData.Set(ssc.SurveyQuestionNumberPrefix + "Experience", "0");

userData.Set(ssc.SurveyQuestionPrefix + "Resolution", "How would you rate the
    resolution we provided");
userData.Set(ssc.SurveyAnswerPrefix + "Resolution", strResolutionAnswer);
userData.Set(ssc.SurveyQuestionNumberPrefix + "Resolution", "1");

userData.Set(ssc.SurveyQuestionPrefix + "Recommend", "Would you recommend us to your
    friends or family");
userData.Set(ssc.SurveyAnswerPrefix + "Recommend", strRecommendAnswer);
userData.Set(ssc.SurveyQuestionNumberPrefix + "Recommend", "2");

userData.Set(ssc.SurveyQuestionPrefix + "Text", "Please provide your overall Experience
    with us");
userData.Set(ssc.SurveyAnswerPrefix + "Text", strCommentsAnswer);
userData.Set(ssc.SurveyQuestionNumberPrefix + "Text", "3");

```

The `contactID` is obtained and added to the collection of user data information along with the survey results:

```

if (strContactID == null || strContactID == "")
{
    try
    {
        InteractionAttributes ia = GetInteractionAttributesByInteractionId(strParentID);
        if (ia != null)
        {
            strContactID = ia.ContactId;
        }
    }
    catch
    {
    }
}

if (strContactID != null && strContactID != "")
    userData.Set(InteractionAttributeListConstants.ContactId, strContactID);
// "ContactId"
userData.Set(InteractionAttributeListConstants.Subject, ssc.SurveySubject);
...

```

The reqSubmit attributes are set and then submitted to Interaction Server with
MediaType=survey:

```

reqSubmit.UserData = userData;
reqSubmit.ParentInteractionId = strParentID;
reqSubmit.MediaType = ssc.SurveyMediatype;
reqSubmit.Queue = ssc.SurveyQueue;
if (strTenantID != null && strTenantID != "")
    reqSubmit.TenantId = int.Parse(strTenantID);
else
    reqSubmit.TenantId = int.Parse(LoadBalancer.getTenantId(ssc.TenantName));
reqSubmit.InteractionType = "Inbound";
reqSubmit.InteractionSubtype = "InboundNew";
reqSubmit.IsOnline = false;
...

```

The appropriate message is displayed to the client:

```

IMessage im = itxProtocol.Request(reqSubmit, new TimeSpan(0, 0, 30));

if (im != null && im.Id == EventAck.MessageId)
{
    EventAck eventAck          = im as EventAck;
    strInteractionID           = eventAck.Extension["InteractionId"].ToString();
    strMessage                  = "Survey successfully submitted to our system with  
reference number " + strInteractionID;
    lblSubmitButton.Visible    = false;
    TextArea_Comments.Enabled  = false;
}

```

```

else if (im != null && im.Id ==
    Genesyslab.Platform.OpenMedia.Protocols.InteractionServer.Events.EventError.MessageId)
{
    Genesyslab.Platform.OpenMedia.Protocols.InteractionServer.Events.EventError
    eventError = im as
    Genesyslab.Platform.OpenMedia.Protocols.InteractionServer.Events.EventError;
    strMessage = "Can't submit survey to our system. Error description: " +
        eventError.ErrorDescription;
}
itxProtocol.Close();
...

```

The `submit_survey()` function submits the survey results of the survey:

```

function submit_survey ()
{
    document.forms[0].Action.value = "Submit";
    document.forms[0].submit();
}

```

E-Mail Sample

This section presents the purpose, functionality overview, and code implementation for the E-mail Sample.

Purpose

The E-mail Sample code demonstrates how a user can send an e-mail request via a web form. The web form e-mail goes through E-mail Server, which routes the e-mail to the appropriate agent using Genesys Universal Routing Server.

The sample files show how to:

- Fill basic e-mail fields via a web form, error-checking for valid user input.
- Connect to E-mail Server using the Platform SDK E-mail Protocol.
- Submit an e-mail request to E-mail Server.
- Report any possible errors in the sample.
- Disconnect from E-mail Server.
- Gather submitted information.
- Communicate with the load-balancing service.
- Communicate with E-mail Server.
- Generate HTML responses and JavaScript code based on responses from E-mail Server.

Functionality Overview

The following sections review the code used in implementing the different e-mail functions:

- “Drawing the Form” on [page 270](#)
- “Declaring and Importing Packages” on [page 272](#)
- “Retrieving Form Data and Constants” on [page 272](#)
- “Processing the Request” on [page 272](#)

Files

The ...\\Email directory contains the E-mail Sample. The sample consists of two files, Email.aspx and Email.aspx.cs.

Code Explanation

The following subsections explain the code in Email.aspx and Email.aspx.cs. As in the other .NET samples, the .aspx file handles most of the page presentation to the user, while the .aspx.cs file contains most of the logic.

User Interface Implementation

The following section outlines the contents of the Email.aspx file.

Drawing the Form

Creating the HTML Header

The Email.aspx file’s code begins by writing some HTML header tags:

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
  <link rel="stylesheet" type="text/css" href="../resource/style/StyleSheet.css"/>
  <title>Genesys Multimedia 8.0.21 Samples. Email.</title>
</head>
```

Handling User Events

The Email.aspx file next provides these JavaScript functions to handle user events:

- window_onload()—has no code. This is an empty function that you can implement.
- Submit_onClick()—sets the action flag to Submit. This function verifies that the E-mail address and Reply from fields are populated and contain acceptable (ASCII) characters. If these conditions are not met, the function presents a dialog box and requests that the user correct the problem. Once the data is acceptable, the function calls the HTML form submit.

- `Reset_onClick()`—calls the form's `reset()` method to clear out all the data entered.

Constructing the HTML Body

Next, the `Email.aspx` code provides some more HTML code to create input boxes. The form `enctype="multipart/form-data"` statement directly below is important to enable the handling of file attachments from the client:

```
<form enctype="multipart/form-data" id="email_form" method="post" action="Email.aspx"
onsubmit="JavaScript:return Submit_onClick (1);" runat="server">
  <table style="width:100%" cellpadding="2">
    <tr>
      <td class="samplesSampleTopToolbar">
        <table >
          <tr >
            <td style="width:617px"
class="samplesSampleNameTitleWhite">Email</td>
            <td style="width:auto" ></td>
          </tr>
        </table>
      </td>
    </tr>
    <tr>
      <td class="samplesSampleParagraph">
        Sample of Email submission over the web based on the PSDK .NET API
        classes and ASPX pages.
      </td>
    </tr>
    <tr>
      <td>
        <table style="width:617px" border="0" >
          <tr>
            <td class="samplesSampleFieldTitle" ></td>
            <td></td>
            <td style="width:6px; vertical-align: middle;"></td>
          </tr>
          <tr>
            <td class="samplesSampleFieldTitle" >First name:</td>
            <td><asp:TextBox CssClass="samplesFieldStyle" ID="FirstName"
AutoPostBack="false" Text="" runat="server" /></td>
            <td style="width:6px; vertical-align: middle;"><font
class="samplesMandatoryMark">*</font></td>
          </tr>...
```

Logic Implementation

The following subsections describe the sample application's logic, which resides primarily in the `Email.aspx.cs` file.

Declaring and Importing Packages

The `Email.aspx.cs` file begins by declaring external packages:

```
using Genesyslab.Platform.Commons.Collections;
using Genesyslab.Platform.Commons.Protocols;
using Genesyslab.Platform.WebMedia.Protocols;
using Genesyslab.Platform.WebMedia.Protocols.EspEmail;
using Genesyslab.Platform.WebMedia.Protocols.EspEmail.Events;
using Genesyslab.Platform.WebMedia.Protocols.EspEmail.Requests;
using Genesyslab.WebApi.Core.ConfigServer;
using Genesyslab.WebApi.Core;
using Genesyslab.Platform.Configuration.Protocols.ConfServer;
using Genesyslab.Platform.Configuration.Protocols.Types;
```

Retrieving Form Data and Constants

The `Email.aspx.cs` file's `Page_Load()` function executes when the application displays pages or the user submits form data. It first defines some hidden fields on these pages, using statements like these:

```
protected void Page_Load(object sender, EventArgs e)
{
    ClientScript.RegisterHiddenField ("Action", "");
    Action.Visible = false;
    strFirstName = FirstName.Text;
    strLastName = LastName.Text;
    strFromAddress = FromAddress.Text;
    strMailBox = Mailbox.Text; ...
}
```

The following statement accesses global constants:

```
SimpleSamplesConstants ssc = new SimpleSamplesConstants ();
```

Processing the Request

Getting Load Balancer and E-Mail Server Instance

This section of the `Email.aspx.cs` code creates a load balancer instance and returns an available instance of E-mail Server for each request. Because e-mail interactions are asynchronous, there is no requirement to tie a particular user to a particular instance of either service. So the instances in this sample are not aliased. The sample code gets new instances of the load balancer and E-mail Server upon the submission of each request.

```

try
{
    ServiceInfo si =
LoadBalancer.GetServiceInfo(CfgAppType.CFGEmailServer,
ssc.TenantName);
    svcHost = si.Host;
    svcPort = si.Port;
    bServiceAvailable = true;
}...

```

If the try-catch block is unsuccessful in getting a handle to E-mail Server, it returns an error message:

```

catch (LoadBalancerException e1)
{
    strErrorMessage = "E-Mail service is not available at this time.
    Please try it later.";
}

```

Preparing the Request

The following code block prepares the request. Note the commented-out logging trigger, which is available for you to implement:

```

if (strAction == "Submit" && bServiceAvailable == true)
{
    Uri emailServerURI = new Uri("tcp://" + svcHost + ":" +
svcPort.ToString());
    Endpoint emailEndPoint = new Endpoint(emailServerURI);

    Genesyslab.Platform.WebMedia.Protocols.EspEmail.EspEmailProtocol
email = new EspEmailProtocol(emailEndPoint);
    //email.EnableLogging(new TraceLogger());
}

```

Filling in the Collection

The following code block fills in a collection with the user data submitted via the form:

```

email.Open();

RequestCreateWebEmailIn request = RequestCreateWebEmailIn.Create();
request.FirstName = strFirstName;
request.LastName = strLastName;
request.FromAddress = strFromAddress;
if (strMailBox != null && strMailBox != "")
    request.Mailbox = strMailBox;
request.Text = strEmailBody;
request.Subject = strSubject;

```

Preparing the Attachment

The following code block prepares an attachment.

```

if (abAttachment != null && abAttachment.Length > 0)
{
    request.Attachments = new EmailAttachmentList();
    EmailAttachment attach = new EmailAttachment();
    attach.Content = abAttachment;
    attach.FileName = AttachmentField.FileName;
    request.Attachments.Add(attach);
}

```

Submitting the Data

This statement submits the data to the server:

```

RequestSubmit reqSubmit = RequestSubmit.Create("email",
mProperties); //was .GetAsKeyValueCollection()

```

Analyzing the Response

Finally, this code block analyzes the server's response, and then reports either a success flag or any error message returned:

```

IMessage im = email.Request (request, new TimeSpan(0, 0, 30));
if (im != null)
{
    if (im.Id == EventCreateWebEmailIn.MessageId)
    {
        EventCreateWebEmailIn eventAck = im as EventCreateWebEmailIn;
        Interaction_Id.Text = eventAck.NewInteractionId;
        strErrorMessage = "Request " + eventAck.NewInteractionId + " has
been successfully submitted to E-mail Server.";
    }
    else if (im.Id == EventError.MessageId)
    {
        EventError eventError = im as EventError;
        strErrorMessage = eventError.FaultString;
    }
}...

```

Genesys 3rd Party Media Sample

This sample demonstrates a web form through which users can create and submit interactions involving attached Genesys 3rd Party Media data (faxes, video, SMS, and so on). The web form provides fields where the user can enter a first name, last name, and three key-value pairs; a drop-down list box where the user can identify the attached media type; and fields that automatically identify the Interaction Server host and port, the Interaction ID, and the workflow queue, as configured for Web API Server .NET's application in Configuration Server.

Warning! For this sample to work reliably, your custom application's code must ensure that each interaction generated and submitted by the application has a unique `InteractionID`. Neither the sample code nor Interaction Server error-check for this requirement, so meeting it is your code's responsibility.

You could choose to allow Interaction Server to generate a unique interaction identifier. In this case, you would design your application to pass an empty `InteractionID`, so that Interaction Server will assign its own `InteractionID` and return that generated ID in the Ack event.

Purpose

The Genesys 3rd Party Media Sample shows how to:

- Connect to Interaction Server using the Interaction API.
- Specify and submit an interaction.
- Update an interaction
- Cancel an interaction.

Functionality Overview

The following sections outline the code used to implement the 3rd Party Media Sample:

- “Creating the HTML Header” on [page 276](#)
- “Constructing the HTML Body” on [page 276](#)
- “Handling User Events” on [page 278](#)
- “Declaring and Importing Packages” on [page 278](#)
- “Declaring Variables and Importing Constants” on [page 279](#)
- “Collecting User Data” on [page 279](#)
- “Drawing the Form and Submitting User Data” on [page 280](#)
- “Processing the Request” on [page 280](#)
- “Spare Event-Handler Prototypes” on [page 281](#)
- “Connecting to Interaction Server” on [page 282](#)
- “Interaction Event Handlers” on [page 283](#)
- “Getting Load Balancer and Interaction Server Instance” on [page 280](#)
- “Submitting Requests” on [page 283](#)
- “Analyzing Responses” on [page 285](#)
- “Closing Connections” on [page 285](#)

Files

The ...\\ItxSubmit directory contains the Genesys 3rd Party Media Sample. The sample consists of two files, ItxSubmit.aspx and ItxSubmit.aspx.cs.

Code Explanation

Like the other .NET samples, the Genesys 3rd Party Media Sample separates user interface and logic components. The following subsections explain the code in ItxSubmit.aspx and ItxSubmit.aspx.cs files.

User Interface Implementation

The ItxSubmit.aspx file controls most of the page presentation to the user. The following subsections explain the code in that file.

Creating the HTML Header

The ItxSubmit.aspx file begins by building an HTML header:

```
<head>
    <link rel="stylesheet" type="text/css"
href= "../Resources/StyleSheet.css"/>
    <title>Genesys Multimedia 8.0.21 Samples. Interaction Server
sample.</title>
    <script src= "../Resources/custom-form-elements.js"
type="text/javascript" ></script>
</head>
```

Constructing the HTML Body

Next, the code includes the HTML code to create the input fields; the Send, Update, and Cancel buttons; and the field that holds messages from the server:

```
<body onload="javascript:window_onload();" class="samplesBody"
onresize="javascript:window_onresize();" >
<script src= "../Resources/helper.js" type="text/javascript" ></script>...

...<tr>
    <td class="samplesSampleParagraph">Submit interaction to Interaction server
with PSDK .NET API classes and ASPX pages<br/><br/></td>
</tr>...

...<td>
    <table border="0">
    <tr>
        <td class="samplesSampleFieldTitle" >Interaction Server host:</td>
        <td><asp:TextBox style="width:550px; vertical-align: middle;"
        CssClass="samplesFieldStyle" id="tbServerName" AutoPostBack="false" Text=""
        runat="server" /></td>
        <td style="width:6px; vertical-align: middle;"><span
        class="samplesMandatoryMark">*</span></td>
```

```

        </tr>
        <tr>
            <td class="samplesSampleFieldTitle" >Interaction Server port:</td>
            <td><asp:TextBox style="width:550px; vertical-align: middle; "
                CssClass="samplesFieldStyle" id="tbPort" AutoPostBack="false" Text="" runat="server"
                /></td>
            <td style="width:6px; vertical-align: middle;"><span
                class="samplesMandatoryMark">*</span></td>
        </tr>...

    <tr>
        <td class="samplesSampleFieldTitle" >Media type:</td>
        <td><asp:TextBox style="width:550px; vertical-align: middle; "
            CssClass="samplesFieldStyle" id="tbMediaType" runat="server"/></td>
        <td style="width:6px; vertical-align: middle;"><span
            class="samplesMandatoryMark">*</span></td>
    </tr>
    <tr>
        <td ></td>
        <td>
            <asp:DropDownList id="selMedia" onchange="selMedia_onchange();"
            runat="server" >
                <asp:ListItem>email</asp:ListItem>
                <asp:ListItem>chat</asp:ListItem>
                <asp:ListItem>callback</asp:ListItem>
                <asp:ListItem>sms</asp:ListItem>
                <asp:ListItem>fax</asp:ListItem>
                <asp:ListItem>imchat</asp:ListItem>
                <asp:ListItem>video</asp:ListItem>
                <asp:ListItem>voice</asp:ListItem>
                <asp:ListItem>voip</asp:ListItem>
                <asp:ListItem>webform</asp:ListItem>
                <asp:ListItem>webcallback</asp:ListItem>
            </asp:DropDownList>
        </td>
        <td style="width:6px; vertical-align: middle;"></td>
    </tr>...

...<tr >
    <td style="width:150px; vertical-align: middle; "
        class="samplesSampleFieldTitle" ></td>
    <td style="width:511px; vertical-align: middle; text-align:right">
        <table>
            <tr align="right">
                <td onclick="javascript:on_itx_submit();"
                class="sampleBlueButton" onmouseover="javascript:MouseOver(this); "
                onmouseout="javascript:MouseOut(this); ">Submit</td>
                <td onclick="javascript:on_itx_stop();"
                class="sampleBlueButton" onmouseover="javascript:MouseOver(this); "
                onmouseout="javascript:MouseOut(this); ">Stop</td>
            </tr>
        </table>
    </td>

```

```

        <td onclick="javascript:on_itx_update();"
class="sampleBlueButton" onmouseover="javascript:MouseOver(this);"
onmouseout="javascript:MouseOut(this);">Update</td>
        <asp:TextBox id="tbCommand" runat="server"></asp:TextBox>
    </tr>
</table>
</td>
    <td style="width:6px; vertical-align: middle;"></td>
</tr>...

```

Handling User Events

Four JavaScript functions handle user events:

- `window_onload()`—called every time the page is loaded. This function currently sets the form's default media type, but you may also modify it to execute any other tasks that should be carried out whenever the page is loaded.
- `selMedia_onChange()`—sets the media type to the value selected by the user.
- `getSelectedOption(opt)`—takes a collection of options for a field and returns the option that has been selected by the user.
- `setSelection(objControl, Value)`—sets the value of a document field (`ObjControl`) to `Value`.

Logic Implementation

The `ItxSubmit.aspx.cs` file contains most of the 3rd Party Media Sample's logic. The following subsections explain the code in that file.

Declaring and Importing Packages

The `ItxSubmit.aspx.cs` file begins by loading external packages into memory:

```

using Genesyslab.Platform.Commons.Collections;
using Genesyslab.Platform.Commons.Protocols;
using Genesyslab.WebApi.Core.ConfigServer;
using Genesyslab.WebApi.Core;
using Genesyslab.Platform.Configuration.Protocols.ConfServer;
using Genesyslab.Platform.OpenMedia.Protocols;
using Genesyslab.Platform.OpenMedia.Protocols.InteractionServer;
using Genesyslab.Platform.OpenMedia.Protocols.InteractionServer.Events;
using Genesyslab.Platform.OpenMedia.Protocols.InteractionServer.Requests;
...
using CfgConnections; //TraceLogger is here

```

Declaring Variables and Importing Constants

Next, the `ItxSubmit.aspx.cs` file declares variables and accesses external constants:

```
public partial class ItxSubmit_ItxSubmit : System.Web.UI.Page
{
    string strHost          = "";
    int iPort               = -1;
    string strInteractionID = "";
    string strMediaType     = "";
    string strQueue         = "";
    int iTenandID           = -1;
    string strFirstName     = "";
    string strLastName      = "";
    bool bAgreeWithRules    = false;
    int iGender             = -1;
    string AttachDataKey1   = "";
    string AttachDataValue1 = "";
    string AttachDataKey2   = "";
    string AttachDataValue2 = "";
    string AttachDataKey3   = "";
    string AttachDataValue3 = "";
    string strMessages      = "";
    bool bConnected         = false;
    InteractionServerProtocol itxProtocol = null;
    SimpleSamplesConstants ssc = new SimpleSamplesConstants();
}
```

Collecting User Data

The Genesys 3rd Party Media Sample consolidates data collection and submission into a discrete function called `CollectSubmitData()`. This function gets invoked by the event handlers that correspond to the three buttons on the UI form (Submit Interaction, Stop Processing, and Update Interaction). This approach avoids duplication of the data collection/submission code.

Here is the code block that performs data collection:

```
protected bool CollectSubmitData()
{
    try
    {
        strHost          = tbServerName.Text;
        strInteractionID = tbInteractionID.Text;
        strMediaType     = tbMediaType.Text;
        strQueue         = selScriptName.Text;
        strFirstName     = tbFirstName.Text;
        strLastName      = tbLastName.Text;
        ...
        AttachDataKey3   = tbAttachDataName1.Text;
        AttachDataValue3 = tbAttachDataValue1.Text;
    }
}
```

```

        strMessages        = "";
        //Try to collect as much as possible. Parse may throw an exception.
        iPort               = int.Parse(tbPort.Text);
        iTenantID           = int.Parse(tbTenantID.Text);
    }
    catch (Exception e)
    {
        strMessages = "Error during submit: \r\n" + e.ToString();
        return false;
    }
    return true;

```

Drawing the Form and Submitting User Data

Next, the `Page_Load()` function initializes the onscreen form by calling the `CollectSubmitData()` function discussed in “[Collecting User Data](#),” above:

```

protected void Page_Load(object sender, EventArgs e)
{
    CollectSubmitData();
    if (strHost == "" || iPort == -1)
    {
        ...
    }
}

```

Processing the Request

Getting Load Balancer and Interaction Server Instance

The next section of code attempts to get instances of the load balancer and Interaction Server. If it succeeds, it captures the Interaction Server host and port information, and the tenant name, to variables:

```

try
{
    ServiceInfo si = LoadBalancer.GetServiceInfo(CfgAppType.CFGInteractionServer,
        ssc.TenantName);
    tbServerName.Text = si.Host;
    tbPort.Text = si.Port.ToString();
}
catch (LoadBalancerException e1)
{
    tbMessages.Text = "Interaction server is not available at this time.
        Please try it later.";
}

try
{
    tbTenantName.Text = ssc.TenantName;
    tbTenantID.Text = LoadBalancer.getTenantId(ssc.TenantName);
}
catch (LoadBalancerException e1)
{
    tbMessages.Text = "Can't find DBID for tenant " + ssc.TenantName + "."; ...
}

```

Loading Endpoints Information This next section of `ItxSubmit.aspx.cs` seeks to load information about interaction-queue endpoints from Configuration Server. This information pertains to the Web API Server application itself, and must be filled in:

```
try
{
    ConfigServerApp ownApp = LoadBalancer.getOwnApp();
    SortedList slOptions = ownApp.Options;

    int iSectionPos = slOptions.IndexOfKey("endpoints:" + tbTenantID.Text);
    if (iSectionPos != -1)
    {
        SortedList slSection = (SortedList)(slOptions.GetByIndex(iSectionPos));
        if (slSection != null)
        {
            for (int i = 0; i < slSection.Count; i++)
            {
                string strReturnKey = (string)(slSection.GetKey(i));
                string strReturnValue = (string)(slSection.GetByIndex(i));
                selScriptName.Items.Add(new ListItem(strReturnKey, strReturnValue));
                ...
            }
        }
        else
        {
            selScriptName.Items.Add(new ListItem("No 'endpoints' section.",
                "bad_configuration"));
        }
        ...
    }
    catch (LoadBalancerException e1)
    {
        tbMessages.Text = "Can't get settings from LoadBalancer.
            Please check your configuration.";
```

Spare Event-Handler Prototypes The next code block defines event handlers that the sample application does not use. These are prototypes for your custom event handlers:

```
private void conn_Opened(object sender, EventArgs e)
{
    bConnected = true;
}

private void conn_Closed(object sender, EventArgs e)
{
    bConnected = false;
}

private void conn_Error(object sender, EventArgs e)
{
    /* remove comments when ErrorEventArgs could be resolved
    ErrorEventArgs e1 = null;
    if (e is ErrorEventArgs)
    e1 = (ErrorEventArgs)e;
```

```
Trace.WriteLine("event Error for mediaServer");
if (e1 != null)
Trace.WriteLine(e1.Cause.StackTrace);
*/}
```

Connecting to Interaction Server

The next section of `ItxSubmit.aspx.cs` attempts to connect to Interaction Server. It connects using the host and port information acquired earlier by the `CollectSubmitData()` function. Unlike the corresponding Java sample (“Interaction Submit (Genesys 3rd Party Media) Sample” on [page 151](#)), this .NET sample does not alias this host/port information.

Warning! The various Web API samples illustrate different ways in which your own applications can communicate with the load balancer and store data from it:

- Acquire a new server for each request.
- Acquire services by stored aliases.
- Pass the destination server’s host and port information, unaliased.

The last option—shown in this sample—is potentially dangerous. It can reveal aspects your network infrastructure (such as your internal server’s name and port) to potential attackers. Therefore, Genesys recommends that you *not* use this technique in any front-end application.

```
protected bool Connect(string host, int port)
{
    try
    {
        Uri itxServerURI = new Uri("tcp://" + host + ":" + port.ToString());
        Endpoint itxEndPoint = new Endpoint(itxServerURI);
        itxProtocol = new InteractionServerProtocol(itxEndPoint);

        itxProtocol.Opened += new EventHandler(conn_Opened);
        itxProtocol.Closed += new EventHandler(conn_Closed);
        itxProtocol.Error += new EventHandler(conn_Error);

        itxProtocol.ClientType = InteractionClient.MediaServer;
        itxProtocol.ClientName = "WebAPIServer8021";

        if (bUseTLS == true)
        {
            KeyValueCollection kvl = new KeyValueCollection();
            kvl[CommonConnection.TlsKey] = 1;
            KeyValueConfiguration cfg = new KeyValueConfiguration(kvl);
            itxProtocol.Configure(cfg);
        }
    }
}
```

```

        itxProtocol.Open();
    }
    catch (Genesyslab.Platform.Commons.Protocols.ProtocolException e1)
    {
        tbMessages.Text = "Can't connect to the Interaction Server.";
        return false;
    }
    catch (Exception e2)
    {
        tbMessages.Text = "Can't connect to the Interaction Server. " + e2.Message;
        return false;
    }
    return true;
}

```

Interaction Event Handlers

The final section of `ItxSubmit.aspx.cs` contains three event handlers, corresponding to the three buttons that the `ItxSubmit.aspx` web form offers to the user:

- `SubmitInteraction_onClick()`
- `StopProcessing_onClick()`
- `UpdateInteraction_onClick()`

Each of these event handlers performs the same basic operations:

- Submit the request selected by the user.
- Analyze the response from Interaction Server, relaying any errors.
- Close the connection to Interaction Server.

These operations are demonstrated below, by code snippets primarily drawn from `SubmitInteraction_onClick()`.

Submitting Requests

The upper try block of `SubmitInteraction_onClick()` submits the user's selected request (in this case, sending the interaction) to Interaction Server. The corresponding catch block traps exceptions raised in the submission attempt. The outer else branch reports an exception if the `CollectSubmitData()` function has failed to capture and parse all required information:

```

protected void SubmitInteraction_onClick(Object sender, EventArgs e)
{
    if (CollectSubmitData() == true)
    {
        try
        {
            if (Connect(strHost, iPort) == true)
            {
                RequestSubmit reqSubmit = RequestSubmit.Create();
                KeyValueCollection userData = new KeyValueCollection();

```

```

        userData.Set(AttachDataKey1, AttachDataValue1);
        userData.Set(AttachDataKey2, AttachDataValue2);
        userData.Set(AttachDataKey3, AttachDataValue3);
        userData.Set("FirstName", strFirstName);
        userData.Set("LastName", strLastName);
        reqSubmit.UserData          = userData;
        reqSubmit.MediaType         = strMediaType;
        reqSubmit.InteractionId     = strInteractionID;
        reqSubmit.Queue             = strQueue;
        reqSubmit.TenantId          = iTenandID;
        reqSubmit.InteractionType   = "Inbound";
        reqSubmit.InteractionSubtype = "InboundNew";
        reqSubmit.IsOnline          = false;

        IMessage im = itxProtocol.Request(reqSubmit, new
                                           TimeSpan(0, 0, 30));

        ...
    catch (Exception exception)
    {
        tbMessages.Text = "Exception occurred.\r\n" +
                          exception.ToString();
    }
}
else
{
    tbMessages.Text = strMessages;
}

```

Warning! The Web API server does not validate dynamic data; that is the responsibility of your application code. In production code corresponding to the code block above, your application should check that names of attached data (such as AttachDataKey1 and AttachDataKey2) are not empty and not identical. In the case of empty or duplicate names, the server might reject the data or userData might throw an exception.

In the StopProcessing_onClick() event handler, the corresponding try block halts processing of the interaction. Note the placeholders in which you can implement your own reason code and description:

```

try
{
    if (Connect(strHost, iPort) == true)
    {
        ReasonInfo reason = ReasonInfo.Create();
        reason.Reason = 123456789;
        reason.ReasonDescription = "Put your reason here";
        RequestStopProcessing requestStopProcessing =
            RequestStopProcessing.Create(strInteractionID, reason);
    }
}

```

```
IMessage im = itxProtocol.Request(requestStopProcessing, new
    TimeSpan(0, 0, 30));
```

In the `UpdateInteraction_onClick()` event handler, the corresponding try block deletes the interaction's third key-value pair, while also updating the first two key-value pairs with new values from the newly submitted data:

```
try
{
    if (Connect(strHost, iPort) == true)
    {
        KeyValueCollection kvcChangedProperties = new KeyValueCollection();
        kvcChangedProperties.Set(AttachDataKey1, AttachDataValue1);
        kvcChangedProperties.Set(AttachDataKey2, AttachDataValue2);
        kvcChangedProperties.Set("FirstName", strFirstName);
        kvcChangedProperties.Set("LastName", strLastName);

        KeyValueCollection kvcDeletedProperties = new KeyValueCollection();
        kvcDeletedProperties.Set(AttachDataKey3, AttachDataValue3);

        requestChangeProperties requestChangeProperties =
            RequestChangeProperties.Create();
        requestChangeProperties.InteractionId = strInteractionID;
        requestChangeProperties.AddedProperties = kvcChangedProperties;
        requestChangeProperties.DeletedProperties = kvcDeletedProperties;
        requestChangeProperties.AddedProperties = new KeyValueCollection();
```

Analyzing Responses The second section of each interaction handler's code analyzes Interaction Server's response, then either reports success or relays any errors. Here is that code block from `SubmitInteraction_onClick()`:

```
if (im != null && im.Id == EventAck.MessageId)
{
    EventAck eventAck = im as EventAck;
    strInteractionID = eventAck.Extension["InteractionId"].ToString();
    tbInteractionID.Text = strInteractionID;
    tbMessages.Text = "Operation successfully submitted to the " + strHost + ":" +
        iPort.ToString();;
}
else if (im != null && im.Id == EventError.MessageId)
{
    EventError eventError = im as EventError;
    tbMessages.Text = "Can't submit interaction to the InteractionServer. " +
        eventError.ErrorDescription;
```

Closing Connections Finally, this line in each interaction handler closes the connection to Interaction Server, to avoid leaking resources:

```
itxProtocol.Close();
```

Stat Server Sample

The `...Statistics` directory contains files for the Stat Server Sample. This sample demonstrates a web form through which users can select from a list of six predefined statistics and retrieve their current value.

Purpose

The Statistics Sample shows how to:

- Connect to Stat Server using the Stat Server API.
- Specify and submit a list of statistics.
- Retrieve a current values of selected statistics.

Functionality Overview

The following sections outline the code used to implement the Statistics Sample:

- “Creating the HTML Header” on [page 287](#)
- “Constructing the HTML Body” on [page 287](#)
- “Declaring and Importing Packages” on [page 288](#)
- “Declaring Variables and Importing Constants” on [page 288](#)
- “Collecting User Data” on [page 289](#)
- “Drawing the Form and Submitting User Data” on [page 289](#)
- “Getting Load Balancer and Stat Server Instance” on [page 290](#)
- “Spare Event-Handler Prototypes” on [page 290](#)
- “Connecting to Stat Server” on [page 291](#)
- “Processing the Request” on [page 290](#)
- “Event Handlers” on [page 292](#)
- “Closing Connections” on [page 294](#)

Files

The `...Statistics` directory contains the Stat Server Sample. The sample consists of two files, `StatInfo.aspx` and `StatInfo.aspx.cs`.

Code Explanation

Like the other .NET samples, the Stat Server Sample separates user interface and logic components. The following subsections explain the code in `StatInfo.aspx` and `StatInfo.aspx.cs` files.

User Interface Implementation

The StatInfo.aspx file controls most of the page presentation to the user. The following subsections explain the code in that file.

Creating the HTML Header

The StatInfo.aspx file begins by building an HTML header:

```
<head>
    <link rel="stylesheet" type="text/css"
href="../Resources/StyleSheet.css"/>
    <title>Genesys Multimedia 8.0.21 Samples. StatServer
sample</title>
</head>
```

Constructing the HTML Body

Next, the code includes the HTML code to create the drop down list that contains statistics for you to display:

```
<body onload="javascript:window.onload();" class="samplesBody">
<script src="../Resources/helper.js" type="text/javascript" ></script>
<script src="../Resources/custom-form-elements.js" type="text/javascript" ></script>

<form id="StatForm" name="StatForm" action="StatInfo.aspx" method="post"
    runat="server">
    <asp:TextBox ID="cmd" runat="server"/>...

...<tr>
    <td class="samplesSampleParagraph">Select predefined statistical information
    below to proceed.</td>
</tr>

    <tr>
        <td>
            <table style="width:617px" border="0">
                <tr>
                    <td class="samplesSampleFieldTitle" ></td>
                    <td></td>
                    <td style="width:6px; vertical-align: middle;"></td>
                </tr>

                <tr>
                    <td style="width:100px" class="samplesSampleFieldTitle">Statistic
name:</td>
                    <td>
                        <asp:DropDownList id="selStatistic" name="selStatistic"
runat="server" >
                            <asp:ListItem Value="1">Chat: total distribution
time</asp:ListItem>
                            <asp:ListItem Value="2">Chat: queue length</asp:ListItem>
```

```

        <asp:ListItem Value="3">Chat: total
distributed</asp:ListItem>
        <asp:ListItem Value="4">Chat: queue-current in
processing</asp:ListItem>
        <asp:ListItem Value="5">Chat: queue-current waiting
processing</asp:ListItem>
        <asp:ListItem Value="6">Webform: total distribution
time</asp:ListItem>
        <asp:ListItem Value="7">Webform: queue length</asp:ListItem>
        <asp:ListItem Value="8">Webform: total
distributed</asp:ListItem>
    </asp:DropDownList>
</td>
    <td style="width:6px; vertical-align: middle;"><font
class="samplesMandatoryMark">*</font></td>
</tr>...

```

Logic Implementation

The StatInfo.aspx.cs file contains most of the Statistics Sample's logic. The following subsections explain the code in that file.

Declaring and Importing Packages

The StatInfo.aspx.cs file begins by loading external packages into memory:

```

using Genesyslab.Platform.Commons.Collections;
using Genesyslab.Platform.Commons.Protocols;
using Genesyslab.WebApi.Core.ConfigServer;
using Genesyslab.WebApi.Core;
using Genesyslab.Platform.Configuration.Protocols.ConfServer;
using System.Threading;
using Genesyslab.Platform.Reporting.Protocols;
using Genesyslab.Platform.Reporting.Protocols.StatServer;
using Genesyslab.Platform.Reporting.Protocols.StatServer.Events;
using Genesyslab.Platform.Reporting.Protocols.StatServer.Requests;
using CfgConnections;    //TraceLogger is here

```

Declaring Variables and Importing Constants

Next, the StatInfo.aspx.cs file declares variables and accesses external constants:

```

public partial class StatInfo : System.Web.UI.Page
{
    string strHost                = "";
    int iPort                    = -1;
    StatServerProtocol statServerProtocol = null;
    SimpleSamplesConstants ssc    =
        new SimpleSamplesConstants();
    string strMessages           = "";
}

```

```

Thread eventThread                = null;
int iSelectedStatistic            = -1;
bool bConnected                   = false;
AutoResetEvent lockStatServerReply =
    new AutoResetEvent(false);
IMessage msgStatInfoReply         = null; ...

```

Collecting User Data

The Stat Server Sample collects user data with the function called `CollectSubmitData()`. Here is the code block that performs data collection:

```

protected bool CollectSubmitData()
{
    try
    {
        if (strHost == "" || iPort == -1)
        {
            SimpleSamplesConstants ssc = new SimpleSamplesConstants();
            try
            {
                ServiceInfo si = LoadBalancer.GetServiceInfo
                    (CfgAppType.CFGStatServer, ssc.TenantName);
                strHost = si.Host;
                iPort = si.Port;
            }
            catch (LoadBalancerException e1)
            {
                tbMessages.Text =
                    "StatServer is not available at this time. " +
                    "Please try it later.";
            }
        }
        iSelectedStatistic = int.Parse(selStatistic.Text);
    }
    catch (Exception e)
    {
        strMessages = "Error during submit: \r\n" + e.ToString();
        return false;
    }
    return true;
}

```

Drawing the Form and Submitting User Data

Next, the `Page_Load()` function initializes the onscreen form by calling the `CollectSubmitData()` function discussed in [“Collecting User Data,”](#) above:

```

protected void Page_Load(object sender, EventArgs e)
{
    CollectSubmitData();
}

```

Processing the Request

Getting Load Balancer and Stat Server Instance

The next section of code attempts to get instances of the load balancer and Stat Server. If it succeeds, it captures the Stat Server host and port information, and the tenant name, to variables:

```
try
{
    if (strHost == "" || iPort == -1)
    {
        SimpleSamplesConstants ssc = new SimpleSamplesConstants();
        try
        {
            ServiceInfo si = LoadBalancer.GetServiceInfo
                (CfgAppType.CFGStatServer, ssc.TenantName);
            strHost = si.Host;
            iPort = si.Port;
        }
        catch (LoadBalancerException e1)
        {
            tbMessages.Text = "StatServer is not available at this time.
                Please try it later.";
        }
    }
}...
```

Spare Event-Handler Prototypes

The next code block defines event handlers that the sample application does not use. These are prototypes for your custom event handlers:

```
private void conn_Opened(object sender, EventArgs e)
{
    bConnected = true;
}

private void conn_Closed(object sender, EventArgs e)
{
    bConnected = false;
}

private void conn_Error(object sender, EventArgs e)
{
    /* remove comments when ErrorEventArgs could be resolved
    ErrorEventArgs e1 = null;
    if (e is ErrorEventArgs)
    e1 = (ErrorEventArgs)e;
    Trace.WriteLine("event Error for mediaServer");
    if (e1 != null)
    Trace.WriteLine(e1.Cause.StackTrace);
    */
}
```

Connecting to Stat Server

The next section of `StatInfo.aspx.cs` attempts to connect to Stat Server. It connects using the host and port information acquired earlier by the `CollectSubmitData()` function.

Warning! The various Web API samples illustrate different ways in which your own applications can communicate with the load balancer and store data from it:

- Acquire a new server for each request.
- Acquire services by stored aliases.
- Pass the destination server's host and port information, unaliased.

The last option—shown in this sample—is potentially dangerous. It can reveal aspects your network infrastructure (such as your internal server's name and port) to potential attackers. Therefore, Genesys recommends that you *not* use this technique in any front-end application.

```
protected bool Connect(string host, int port)
{
    Uri statServerURI = new Uri("tcp://" + host + ":" +
        + port.ToString());
    Endpoint statEndPoint = new Endpoint(statServerURI);
    statServerProtocol = new StatServerProtocol(statEndPoint);
    statServerProtocol.EnableLogging
        (new TraceLogger(TraceLogger.LevelDebug));
    statServerProtocol.Opened += new EventHandler(conn_Opened);
    statServerProtocol.Closed += new EventHandler(conn_Closed);
    statServerProtocol.Error += new EventHandler(conn_Error);
    statServerProtocol.ClientId = 777;
    statServerProtocol.ClientName = "WebAPIServerDotNet";
    eventThread = new Thread(new ThreadStart(StatServerEventThread));
    eventThread.Start();

    try
    {
        statServerProtocol.Open();
    }
    catch (Genesyslab.Platform.Commons.Protocols.ProtocolException e1)
    {
        tbMessages.Text = "Can't connect to the StatServer.";
        return false;
    }
    return true;
}
```

The `Connect(string, int)` method above, uses the `StatServerEventThread` thread to handle responses from StatServer and to notify the main thread about these responses by setting the state of the `lockStatServerReply` object. The

main thread waits for these notifications and analyses the response or exits by time-out.

Event Handlers

The final section of StatInfo.aspx.cs contains the GetStatInfo_onClick() method:

The GetStatInfo_onClick() determines which statistic the user has selected from the drop-down list, and calls the RequestStatInfo() method to submit the selected request to Stat Server and return the resulting value.

Here is the GetStatInfo_onClick() method:

```
protected void GetStatInfo_onClick(Object sender, EventArgs e)
{
    if (CollectSubmitData() == true)
    {
        try
        {
            if (Connect(strHost, iPort) == true)
            {
                string strResult = "";
                switch (iSelectedStatistic)
                {
                    case 1:
                        strResult = RequestStatInfo
                            (ssc.TenantName, ssc.ChatQueue,
                             "eserviceinteractionstat.jar:cs
                             total distribution time chat",
                             StatisticType.Historical);
                        break;
                    case 2:
                        strResult = RequestStatInfo
                            (ssc.TenantName, ssc.ChatQueue,
                             "eserviceinteractionstat.jar:cs
                             queue length chat",
                             StatisticType.Current);
                        break;
                    case 3:
                        strResult = RequestStatInfo
                            (ssc.TenantName, ssc.ChatQueue,
                             "eserviceinteractionstat.jar:cs
                             total distributed chat",
                             StatisticType.Historical);
                        break;
                    case 4: ...
                    default:
                        strResult = "Incorrect selected option.";
                        break;
                }
                statServerProtocol.Close();
            }
        }
    }
}
```

```

        strMessages = strHost + ":" + iPort.ToString() + "\r\n";
        strMessages += "Stat result = " + strResult + "\r\n";
    }
}
catch (Exception exception)
{
    strMessages =
        "Exception occurred.\r\n" + exception.ToString();
}
}
tbMessages.Text = strMessages; ...

```

The code for the RequestStatInfo() is shown below:

```

public string RequestStatInfo(string strTenantName, string strQueueName,
    string strStatMetrics, StatisticType stType)
{
    string strReturnValue = "";
    StatisticObject objectDescription =
        new StatisticObject(strTenantName, strQueueName, StatisticObjectType.StagingArea);
    StatisticMetric statisticMetric = new StatisticMetric(strStatMetrics);
    statisticMetric.TimeProfile = "CollectorDefault";
    statisticMetric.TimeRange = "Range0-120";
    Statistic queueStat = new Statistic(objectDescription, statisticMetric);
    StatisticsCollection statisticsCollection = new StatisticsCollection();
    statisticsCollection.AddStatistic(queueStat);

    Notification notification = Notification.Create
        (NotificationMode.Periodical, 100000);
    //Notification notification = Notification.Create(NotificationMode.Immediate, 1);
    RequestOpenPackage requestOpenPackage = RequestOpenPackage.Create
        (12345, stType, statisticsCollection, notification);
    IMessage m = statServerProtocol.Request(requestOpenPackage);

    if (m != null)
    {
        if (m.Name == EventPackageOpened.MessageName)
        {
            if (false == lockStatServerReply.WaitOne(15000, true))
            {
                strReturnValue = "Timeout occurred. No response from StatServer.";
            }
        }
        else if (msgStatInfoReply != null &&
            msgStatInfoReply.Name == EventPackageInfo.MessageName)
        {
            EventPackageInfo epi = msgStatInfoReply as EventPackageInfo;
            IEnumerator Enumerator = epi.Statistics.GetEnumerator();
            while (Enumerator.MoveNext())
            {
                Statistic stat = Enumerator.Current as Statistic;
            }
        }
    }
}

```

```

        strReturnValue += stat.StringValue;
        break;
    }
}
else
{
    if (msgStatInfoReply != null)
        strReturnValue = "Error.Unexpected message from StatServer: "
            + msgStatInfoReply.Name;
    else
        strReturnValue = "Error. Empty message from StatServer.";
}
}
else if (m.Name == EventPackageError.MessageName)
{
    EventPackageError epe = m as EventPackageError;
    strReturnValue = epe.Description;
}
else
{
    strReturnValue = "Error. Unexpected message from StatServer: " + m.Name;
}
}
else
{
    strReturnValue = "Error. Empty message from StatServer.";
}
}
return strReturnValue;
}

```

Closing Connections Finally, this line in each interaction handler closes the connection to Stat Server, to avoid leaking resources:

```
statServerProtocol.Close();
```

Universal Contact Server Sample

This sample demonstrates a web form through which users can access a given contact's interaction history from the `Interaction` table of Universal Contact Server's database and display it in an ordered list.

Purpose

The Universal Contact Server Sample shows how to:

- Connect to Universal Contact Server using the UCS API.
- Query the database based on first name, last name and e-mail address.

- Displays the result of the query.
- Disconnect from the Universal Contact Server.

Functionality Overview

The following sections outline the code used to implement the Universal Contact Server Sample:

- “Creating the HTML Header” on [page 295](#)
- “Constructing the HTML Body” on [page 296](#)
- “Declaring and Importing Packages” on [page 298](#)
- “Declaring Variables and Importing Constants” on [page 298](#)
- “Collecting User Data” on [page 299](#)
- “Drawing the Form and Submitting User Data” on [page 299](#)
- “Getting Load Balancer and UCS Instance” on [page 300](#)
- “Connecting to Interaction Server” on [page 300](#)
- “Closing Connections” on [page 308](#)

Files

The ...\\UCS directory contains the Universal Contact Server Sample. The sample consists of four files, `UCS.aspx`, `UCS.aspx.cs`, `Action.aspx`, and `Action.aspx.cs`.

Code Explanation

Like the other .NET samples, the Universal Contact Server Sample separates user interface and logic components. The following subsections explain the code in the `UCS.aspx`, `UCS.aspx.cs`, `Action.aspx`, and `Action.aspx.cs` files.

User Interface Implementation

The `UCS.aspx` file controls most of the page presentation to the user. The following subsections explain the code in that file.

Creating the HTML Header

The `UCS.aspx` file begins by building an HTML header:

```
<head>
  <link rel="stylesheet" type="text/css"
href="../../Resources/StyleSheet.css"/>
  <title>Genesys Multimedia 8.0.21 Samples. UCS.</title>
</head>
```

Constructing the HTML Body Next, the code includes the HTML code to create the input fields for first name, last name, and e-mail address; the Search button; and to display the query results:

```
<form onsubmit="javascript:return SearchButton_onClick();" id="ucs_form"
name="ucs_form" method="post" action="UCS.aspx" runat="server">
  <asp:TextBox ID="cmd" runat="server"/>
  <table width="100%" cellpadding="2">
    <tr>
      <td class="samplesSampleTopToolbar">
        <table>
          <tr>
            <td style="width:617px" class="samplesSampleNameTitleWhite">UCS
Service Sample</td>
            <td style="width:auto" ></td>
          </tr>
        </table>
      </td>
    </tr>

    <tr>
      <td class="samplesSampleParagraph">Please enter your information below to
proceed.</td>
    </tr>

    <tr>
      <td>
        <table style="width:617px" border="0">
          <tr>
            <td class="samplesSampleFieldTitle" ></td>
            <td></td>
            <td style="width:6px; vertical-align: middle;"></td>
          </tr>...

          ...<tr>
            <td class="samplesSampleFieldTitle" >First name:</td>
            <td><asp:TextBox CssClass="samplesFieldStyle" ID="tbFirstName"
AutoPostBack="false" Text="" runat="server" /></td>
            <td style="width:6px; vertical-align: middle;"><font
class="samplesMandatoryMark">*</font></td>
          </tr>

          <tr>
            <td class="samplesSampleFieldTitle" >Last name:</td>
            <td><asp:TextBox CssClass="samplesFieldStyle" ID="tbLastName"
AutoPostBack="false" Text="" runat="server" /></td>
            <td style="width:6px; vertical-align: middle;"><font
class="samplesMandatoryMark">*</font></td>
          </tr>
```

```

        <tr>
            <td class="samplesSampleFieldTitle" >E-mail:</td>
            <td><asp:TextBox CssClass="samplesFieldStyle" ID="tbEMail"
AutoPostBack="false" Text="" runat="server" /></td>
            <td style="width:6px; vertical-align: middle;"><font
class="samplesMandatoryMark">*</font></td>
        </tr>

        <tr>
            <td class="samplesSampleFieldTitle"></td>
            <td align="right">
                <table>
                    <tr>
                        <td runat="server" onclick="javascript:submit_onclick();"
class="sampleBlueButton" onmouseover="javascript:MouseOver(this);"
onmouseout="javascript:MouseOut(this);">Submit</td>
                    </tr>
                </table>
            </td>
            <td style="width:6px; vertical-align: middle;"></td>
        </tr>

        <tr>
            <td colspan="3" class="samplesSampleParagraph">
                Color diagram:
            </td>
        </tr>

        <tr>
            <td colspan="3">
                <table style="width:100%">
                    <tr>
                        <td class="samplesSampleFieldTitleFontOnly" style="width:25%;
background-color:<%=threadBackgroundColorNew%" align="center">New messages</td>
                        <td class="samplesSampleFieldTitleFontOnly" style="width:25%;
background-color:<%=threadBackgroundColorOld%" align="center">No new messages</td>
                        <td class="samplesSampleFieldTitleFontOnly" style="width:25%;
background-color:<%=emailFromCustomerColor%" align="center">Message from
customer</td>
                        <td class="samplesSampleFieldTitleFontOnly" style="width:25%;
background-color:<%=emailFromAgentColor%" align="center">Message from call
center</td>
                    </tr>
                </table>
            </td>
        </tr>
        <tr>
            <td colspan="4">
                <hr style="width:617px" class="samplesHrSeparator" />
            </td>
        </tr>
    </table>
</td>

```

```

        </tr>

        <tr>
            <td colspan="3"><asp:Label ID="ResultData"
runat="server"></asp:Label></td>
        </tr>...

```

Logic Implementation

The UCS.aspx.cs file contains most of the Universal Contact Server Sample's logic. The following subsections explain the code in that file.

Declaring and Importing Packages

The UCS.aspx.cs file begins by loading external packages into memory:

```

using System;
using System.Data;
using System.Text;
using System.Configuration;
using System.Collections;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using Genesyslab.Platform.Commons.Collections;
using Genesyslab.Platform.Commons.Protocols;
using Genesyslab.WebApi.Core.ConfigServer;
using Genesyslab.WebApi.Core;
using Genesyslab.Platform.Configuration.Protocols.ConfServer;
using Genesyslab.Platform.Contacts.Protocols.ContactServer;
using Genesyslab.Platform.Contacts.Protocols.ContactServer.Requests;
using Genesyslab.Platform.Contacts.Protocols.ContactServer.Events;
using Genesyslab.Platform.Contacts.Protocols;
using Genesyslab.Platform.Contacts;
using CfgConnections;...
using CfgConnections; //TraceLogger is here

```

Declaring Variables and Importing Constants

Next, the UCS.aspx.cs file declares variables and accesses external constants:

```

public string threadColorNew           = "#8CB388";
public string threadBackgroundColorNew = "#CDDECB";
public string threadColorOld           = "#C0C0C0";
public string threadBackgroundColorOld = "#E6E6E6";
public string baseBoldColor            = "#000080";
public string emailFromCustomerColor  = "#EFF1F5";

```

```

public string emailFromCustomerBorderColor = "#DBE1EA";
public string emailFromAgentColor         = "#FAFAF5";
public string emailFromAgentBorderColor   = "#EAE8DB";
public string bodyColor                   = "#FFFFFF";
public string bodyBackgroundColor         = "#000000";
string strFirstName                       = "";
string strLastName                        = "";
string strEmail                           = "";
string strMessage                         = "";
string strTableContent                    = "";
public string strContactID                 = "";
string strHost                            = "";
int iPort                                 = -1;
UniversalContactServerProtocol ucsp       = null;
SimpleSamplesConstants ssc                 = null;;

```

Collecting User Data

The Universal Contact Server Sample collects user data with the function called `CollectSubmitData()`. Here is the code block that performs data collection:

```

protected bool CollectSubmitData()
{
    try
    {
        strHost      = tbServerName.Text;
        strFirstName = tbFirstName.Text;
        strLastName  = tbLastName.Text;
        strEmail     = tbEmail.Text;
        iPort        = int.Parse(tbPort.Text); ;
    }
    catch (Exception e)
    {
        strMessage = "Error during submit: \r\n" + e.ToString();
        return false;
    }
    return true;
};

```

Drawing the Form and Submitting User Data

Next, the `Page_Load()` function initializes the onscreen form by calling the `CollectSubmitData()` function discussed in [“Collecting User Data,”](#) above:

```

protected void Page_Load(object sender, EventArgs e)
{
    CollectSubmitData();
    ssc = new SimpleSamplesConstants();
    if (strHost == "" || iPort == -1)
    {...

```

Processing the Request

Getting Load Balancer and UCS Instance

The next section of code attempts to get instances of the load balancer and Universal Contact Server. If it succeeds, it captures the Interaction Server host and port information, and the tenant name, to variables:

```
try
{
    ServiceInfo si = LoadBalancer.GetServiceInfo
        (CfgAppType.CFGContactServer, ssc.TenantName);
    tbServerName.Text = si.Host;
    tbPort.Text = si.Port.ToString();
}
catch (LoadBalancerException e1)
{
    ResultData.Text = "Contact server is not available at this time.
        Please try it later.";
}...
```

Connecting to Interaction Server

The next section of UCS.aspx.cs attempts to connect to Universal Contact Server. It connects using the host and port information acquired earlier by the CollectSubmitData() function.

```
protected void Search_onClick(Object sender, EventArgs e)
{
    CollectSubmitData();
    try
    {
        Uri ucsURI = new Uri("tcp://" + strHost + ":"
            + iPort.ToString());
        Endpoint ucsEndPoint = new Endpoint(ucsURI);
        ucsp = new UniversalContactServerProtocol(ucsEndPoint);
        ucsp.EnableLogging(new TraceLogger(TraceLogger.LevelDebug));
        ucsp.Opened += new EventHandler(conn_Opened);
        ucsp.Closed += new EventHandler(conn_Closed);
        ucsp.Error += new EventHandler(conn_Error);
        ucsp.Open();...
```

The next part of the code submits the users query information, displays the sorted results, and closes the connection to the UCS:

```
IMessage msg = ucsp.Request(CreateGetContacts(true));
if (msg != null && msg.GetType() == typeof(EventGetContacts))
{
    EventGetContacts eventGC = msg as EventGetContacts;
    ContactDataList cdl = eventGC.ContactData;
    if (cdl != null && (int)eventGC.CurrentCount > 0)
    {
        Contact contact = cdl.Get(0);
        strContactID = contact.Id;
```

```

msg = ucsp.Request(CreateGetInteractions(strContactID));
if (msg != null && msg.GetType() == typeof(EventError))
{
    EventError error = msg as EventError;
    error.FaultString = error.FaultString;
}
else if (msg != null && msg.GetType() == typeof
(EventGetInteractionsForContact))
{
    EventGetInteractionsForContact egifc =
        msg as EventGetInteractionsForContact;
    ContactInteractionList cil = egifc.ContactInteractions;
    cil.Sort(new ContactInteractionsComparer());
    strTableContent = BuildInteractionTable(cil);
}...
..ucsp.Close();
ResultData.Text = strTableContent;...

```

The `CreateGetContacts(bool)` method is called by `Search_onClick` to retrieve the list of sorted contacts:

```

private IMessage CreateGetContacts(bool whisSort)
{
    RequestGetContacts gcr      = new RequestGetContacts();
    ComplexSearchCriteria csc   = null;
    gcr.TenantId                = int.Parse(LoadBalancer.getTenantId
        (ssc.TenantName));
    gcr.MaxCount                 = 5;
    gcr.Restricted               = false;
    gcr.SearchCriteria           = new SearchCriteriaCollection();

    if (strFirstName != "")
    {
        SimpleSearchCriteria simple1 = new SimpleSearchCriteria();
        simple1.AttrName = ContactSearchCriteriaConstants.FirstName;
        simple1.AttrValue = strFirstName;
        simple1.Operator = Operators.Equal;

        csc = new ComplexSearchCriteria();
        csc.Prefix = Prefixes.And;
        csc.Criteria = new SearchCriteriaCollection();
        csc.Criteria.Add(simple1);
        gcr.SearchCriteria.Add(csc);
    }

    if (strLastName != "")
    {
        SimpleSearchCriteria simple2 = new SimpleSearchCriteria();
        simple2.AttrName = ContactSearchCriteriaConstants.LastName;
        simple2.AttrValue = strLastName;...
    }
}

```

```

...if (whisSort)
{
    gcr.SortCriteria = new SortCriteriaCollection();
    SortCriteria srt = new SortCriteria();
    srt.SortIndex = 0;
    srt.AttrName = ContactSortCriteriaConstants.FirstName;
    srt.SortOperator = SortMode.Ascending;
    gcr.SortCriteria.Add(srt);
}
return gcr; ...

```

The `CreateGetInteractions(string)` method is called by `Search_onClick` to retrieve the list of sorted interactions:

```

private IMessage CreateGetInteractions(string strContactID)
{
    RequestGetInteractionsForContact rgifc =
        new RequestGetInteractionsForContact();
    rgifc.ContactId = strContactID;
    rgifc.AttributeList = new StringList();
    rgifc.AttributeList.
        Add(InteractionAttributeListConstants.Id);
    rgifc.AttributeList.
        Add(InteractionAttributeListConstants.TypeId);
    rgifc.AttributeList.
        Add(InteractionAttributeListConstants.SubTypeId);
    rgifc.AttributeList.
        Add(InteractionAttributeListConstants.MediaTypeId); ...

    ...rgifc.SearchCriteria = new SearchCriteriaCollection();
    SimpleSearchCriteria searchByMediaType =
        new SimpleSearchCriteria();
    searchByMediaType.Operator = Operators.Equal;

    searchByMediaType.AttrName =
        InteractionSearchCriteriaConstants.MediaTypeId;
    searchByMediaType.AttrValue = "email";
    rgifc.SearchCriteria.Add(searchByMediaType);

    rgifc.SortCriteria = new SortCriteriaCollection();

    SortCriteria srt1 = new SortCriteria();
    srt1.AttrName = InteractionSortCriteriaConstants.ThreadId;
    srt1.SortIndex = 0;
    srt1.SortOperator = SortMode.Ascending;
    rgifc.SortCriteria.Add(srt1);

    SortCriteria srt2 = new SortCriteria();
    srt2.AttrName = InteractionSortCriteriaConstants.StartDate;
    srt2.SortIndex = 1;
}

```

```
srt2.SortOperator = SortMode.Descending;
rgifc.SortCriteria.Add(srt2);...
```

The BuildInteractionTable(ContactInteractionList)

method is called by Search_onClick to create a table of sorted interactions:

```
public string BuildInteractionTable(ContactInteractionList cil)
{
    string strEmailColor          = "";
    string strEmailBorderColor    = "";
    string strLastThreadID        = "";
    string strCurrentThreadID     = "";
    string strSubtypeId           = "";
    string strMediaTypeId         = "";
    string strTypeId              = "";
    string strSubject              = "";
    string strWebSafeEmailStatus  = "";
    int iThreadCounter            = 0;
    string strInteractionState     = "New";
    string strThreadInteractionState = "Read";
    bool bFromCustomer            = false;
    StringBuilder strTableContent = new StringBuilder ("<table
border=\"0\" width=\"100%\" id=\"threadsTable\" cellpadding=\"0\"
style=\"border-collapse: collapse\"><tr>
    int iTotalIncluded            = 0;

    for (int i = 0; i < cil.Count; i++)
    {
        ContactInteraction ci    = cil.Get (i);
        Hashtable htAttributes   =
GetAttributesAsHashtable(ci.InteractionAttributes);
        strMediaTypeId           = GetAttributeAsString(htAttributes,
InteractionAttributeListConstants.MediaTypeId);
        strTypeId                = GetAttributeAsString(htAttributes,
InteractionAttributeListConstants.TypeId);
        strSubtypeId             = GetAttributeAsString(htAttributes,
InteractionAttributeListConstants.SubtypeId);
        strWebSafeEmailStatus    = GetAttributeAsString(htAttributes,
InteractionAttributeListConstants.WebSafeEmailStatus);

        if (strSubtypeId == "OutboundAutoResponse" || strSubtypeId == "OutboundNew"
|| strSubtypeId == "OutboundNotification" || strSubtypeId == "OutboundReply")
            bFromCustomer = false;
        else
            bFromCustomer = true;

        if (!bFromCustomer)
        {
            strInteractionState = "New";
            if (strWebSafeEmailStatus != "")
```

```

        strInteractionState = strWebSafeEmailStatus;
    }
    else
        strInteractionState = "Read";

```

The following conditional statement will allow you to display only e-mail interactions retrieved from UCS:

```

if (strMediaTypeId == "email" && (strSubtypeId == "InboundNew"
    || strSubtypeId == "OutboundAutoResponse"
    || strSubtypeId == "InboundCustomerReply" || strSubtypeId == "OutboundNew"
    || strSubtypeId == "OutboundNotification" || strSubtypeId == "OutboundReply"))
{
    iTotalIncluded++;
    strCurrentThreadID = GetAttributeAsString
        (htAttributes, InteractionAttributeListConstants.ThreadId);
    strSubject = GetAttributeAsString
        (htAttributes, InteractionAttributeListConstants.Subject);

    if (strCurrentThreadID != strLastThreadID)
    {
        if (iThreadCounter > 0)
        {
            strTableContent.Append("
                                </table></td></tr></table>\r\n");
            if (strThreadInteractionState == "New")
            {
                strTableContent.Append(" <script language=\"JavaScript\">\r\n");
                strTableContent.Append("  ChangeThreadStatus(" + iThreadCounter + ", '"
                    + threadColorNew + "', '" + threadBackgroundColorNew + "');\r\n");
                strTableContent.Append(" </script>\r\n");
            }
            strTableContent.Append("
                                </td></tr>\r\n");
        }
        ...
    }
}

```

The Action.aspx.cs file contains the code needed to allow the user to mark the e-mail as read, print the thread, or print the e-mail. These actions are made possible by using the CreateInteractionUpdateAttributes, CreateGetInteractionsForThreadId, and CreateGetInteractionsByEmailId methods. These methods are called by the Page_Load method:

```

protected void Page_Load(object sender, EventArgs e)
{
    ...if (strAction == "mark_as_read")
    {
        msg = ucsp.Request(CreateInteractionUpdateAttributes(strEmail_Id, iTenant));

        if (msg != null && msg.GetType() == typeof(EventUpdateInteraction))
        {
            strTableContent = new StringBuilder("<h2 align=\"center\">

```

```

        Email has been marked as read.</h2>");
        strTableContent.Append
        ("<a align=\"center\"href=\"JavaScript:window.opener.RefreshTable();close();\">
        Close this window and refresh e-mails history.</a>");
    }
    else if (msg == null)
    {
        strTableContent = new StringBuilder("<h2 align=\"center\">
        Empty response from UCS. Please check UCS log for details.</h2>");
    }
}
else if (strAction == "print_thread")
{
    msg = ucsp.Request(CreateGetInteractionsForThreadId(strContact_Id, strThread_Id));
    strTableContent = new StringBuilder("<h2 align=\"center\">
    Email thread history</h2>\r\n");
}
else if (strAction == "print_email")
{
    msg = ucsp.Request(CreateGetInteractionsByEmailId(strContact_Id, strEmail_Id));
    strTableContent = new StringBuilder("<h2 align=\"center\">Email info</h2>\r\n");
}...

```

The CreateInteractionUpdateAttributes(string, int) method updates the status of an e-mail to read:

```

private IMessage CreateInteractionUpdateAttributes(string strInteractionID,
                                                    int iTenantID)
{

```

The code below sets the mandatory properties of RequestUpdateInteraction object:

```

RequestUpdateInteraction rui          = new RequestUpdateInteraction();
rui.InteractionAttributes              = new InteractionAttributes();
rui.InteractionAttributes.Id          = strInteractionID;
rui.InteractionAttributes.TenantId    = iTenantID;
rui.InteractionAttributes.EntityTypeId = EntityTypes.EmailOut;
rui.EntityAttributes                  = new EmailOutEntityAttributes();
rui.InteractionAttributes.OtherFields = new KeyValueCollection();

```

Set the value of attribute that we want to update:

```

rui.InteractionAttributes.OtherFields.
    Add(InteractionAttributeListConstants.WebSafeEmailStatus, "read");
return rui;
}

```

The CreateGetInteractionsForThreadId(string, string) method creates an e-mail history list for a given interaction:

```
private IMessage CreateGetInteractionsForThreadId(string strContactID,
                                                  string strThreadId)
{
    RequestGetInteractionsForContact rgifc = new RequestGetInteractionsForContact();
    rgifc.ContactId = strContactID;
    rgifc.AttributeList = new StringList();
```

Set the list of attributes that we want to retrieve from UCS:

```
rgifc.AttributeList.Add(InteractionAttributeListConstants.Id);
rgifc.AttributeList.Add(InteractionAttributeListConstants.TypeId);
rgifc.AttributeList.Add(InteractionAttributeListConstants.SubtypeId);
rgifc.AttributeList.Add(InteractionAttributeListConstants.MediaTypeId);
rgifc.AttributeList.Add(InteractionAttributeListConstants.Subject);
rgifc.AttributeList.Add(InteractionAttributeListConstants.Text);
rgifc.AttributeList.Add(InteractionAttributeListConstants.ThreadId);
rgifc.AttributeList.Add(InteractionAttributeListConstants.WebSafeEmailStatus);
rgifc.AttributeList.Add(InteractionAttributeListConstants.StartDate);
```

Note: Requesting the `InteractionAttributeListConstants.IxnAttributes` attribute may result in a dramatic decrease in the performance of UCS. Request it only if it is required.

```
//rgifc.AttributeList.Add(InteractionAttributeListConstants.IxnAttributes);

rgifc.SearchCriteria = new SearchCriteriaCollection();
```

Search for only interaction that meet the search criteria, ThreadID:

```
SimpleSearchCriteria searchByThreadId = new SimpleSearchCriteria();
searchByThreadId.Operator = Operators.Equal;
searchByThreadId.AttrName = InteractionSearchCriteriaConstants.ThreadId;
searchByThreadId.AttrValue = strThreadId;
rgifc.SearchCriteria.Add(searchByThreadId);

rgifc.SortCriteria = new SortCriteriaCollection();
```

Sort the Interactions by ThreadId and then by StartDate:

```
SortCriteria srt1 = new SortCriteria();
srt1.AttrName = InteractionSortCriteriaConstants.ThreadId;
srt1.SortIndex = 0;
srt1.SortOperator = SortMode.Ascending;
rgifc.SortCriteria.Add(srt1);

SortCriteria srt2 = new SortCriteria();
srt2.AttrName = InteractionSortCriteriaConstants.StartDate;
srt2.SortIndex = 1;
srt2.SortOperator = SortMode.Ascending;
```

```

    rgifc.SortCriteria.Add(srt2);

    return rgifc;
}

```

The `CreateGetInteractionsByEmailId(string, string)` method allows you to print a specific e-mail:

```

private IMessage CreateGetInteractionsByEmailId(string strContactID, string strEmailID)
{
    RequestGetInteractionsForContact rgifc = new RequestGetInteractionsForContact();
    rgifc.ContactId = strContactID;
    rgifc.AttributeList = new StringList();

```

Set the list of attributes that we want to retrieve from UCS:

```

    rgifc.AttributeList.Add(InteractionAttributeListConstants.Id);
    rgifc.AttributeList.Add(InteractionAttributeListConstants.TypeId);
    rgifc.AttributeList.Add(InteractionAttributeListConstants.SubtypeId);
    rgifc.AttributeList.Add(InteractionAttributeListConstants.MediaTypeId);
    rgifc.AttributeList.Add(InteractionAttributeListConstants.Subject);
    rgifc.AttributeList.Add(InteractionAttributeListConstants.Text);
    rgifc.AttributeList.Add(InteractionAttributeListConstants.ThreadId);
    rgifc.AttributeList.Add(InteractionAttributeListConstants.WebSafeEmailStatus);
    rgifc.AttributeList.Add(InteractionAttributeListConstants.StartDate);

```

Note: Requesting the `InteractionAttributeListConstants.IxnAttributes` attribute may result in a dramatic decrease in the performance of UCS. Request it only if it is required.

```

//rgifc.AttributeList.Add(InteractionAttributeListConstants.IxnAttributes);

    rgifc.SearchCriteria = new SearchCriteriaCollection();
    SimpleSearchCriteria searchByThreadId = new SimpleSearchCriteria();
    searchByThreadId.Operator = Operators.Equal;
    searchByThreadId.AttrName = InteractionSearchCriteriaConstants.Id;
    searchByThreadId.AttrValue = strEmailID;

```

Search the specific e-mail by the e-mail's ID:

```

    rgifc.SearchCriteria.Add(searchByThreadId);

    rgifc.SortCriteria = new SortCriteriaCollection();

```

To ensure that your results are still sorted, sort the Interactions by `ThreadId` and then by `StartDate`:

```

SortCriteria srt1 = new SortCriteria();
srt1.AttrName = InteractionSortCriteriaConstants.ThreadId;
srt1.SortIndex = 0;
srt1.SortOperator = SortMode.Ascending;
rgifc.SortCriteria.Add(srt1);

SortCriteria srt2 = new SortCriteria();
srt2.AttrName = InteractionSortCriteriaConstants.StartDate;
srt2.SortIndex = 1;
srt2.SortOperator = SortMode.Descending;
rgifc.SortCriteria.Add(srt2);

return rgifc;
}

```

Closing Connections

Finally, this line closes the connection to Universal Contact Server, to avoid leaking resources:

```
ucsp.Close();
```

Cobrowse Samples Overview

This section outlines files common to all the Cobrowse Samples.

Common Files

The Basic Cobrowse, the Cobrowse with Initial Startup Page, the Cobrowse with Meet Me, and the Chat and Cobrowse samples all use certain common files:

- `hbmessaging.js`—A JavaScript file that contains the API for Cobrowsing Server.
- `qstring.js`—A JavaScript file that contains a utility class.
- `hbmessage_to_var.js`—A messaging file that provides messages from the cobrowse frame to the web-application frame.
- `blank.html`—The default blank page for initializing empty frames that are used by the API.
- `hbapi.html`—Supports the client-side API and the cobrowse applet.
- `hbmessagingform.html`—Increases security by sending agent login information as an HTTP POST request, instead of as a GET request.
- `hbmessage_to_var.html`—A messaging file that provides messages from the cobrowse frame to the web-application frame.

Warning! The `qstring.js`, `hbmessage_to_var.js`, and `hbmessage_to_var.html` files must all reside in the same directory.

For background information about the purposes of these common files, refer to the KANA Response Live documentation that is listed in “Related Documentation Resources” on [page 363](#). Genesys licenses certain Response Live (formerly Hipbone) cobrowsing components from KANA Software, Inc.

Basic Cobrowse Sample

The `...\CoBrowse` directory contains the files for the Basic Cobrowse Sample. This sample demonstrates Basic Cobrowse functionality.

Purpose

The Cobrowse Sample code demonstrates basic cobrowse functionality that uses the Cobrowsing Server API.

Functionality Overview

The following sections outline the code that is used to implement the Basic Cobrowse Sample:

- “Getting Instances of Load Balancer and Cobrowse Server” on [page 310](#)
- “Getting the Conavigation Channel ID” on [page 310](#)
- “Drawing the Form” on [page 310](#)
- “Declarations” on [page 312](#)
- “Event Handlers” on [page 313](#)

Files

The `...\CoBrowse` directory contains the Basic Cobrowse Sample. This sample includes three files beyond those that are covered in the preceding “Common Files” on [page 308](#) section:

- `CoBrowse.htm`—Sets up the display frame.
- `CoBrowseEventHandler.aspx` and `CoBrowseEventHandler.aspx.cs`—Contain the logic of the sample.

Code Explanation

The following subsections explain the code that appears in the `CoBrowseEventHandler.aspx.cs` file. This file contains two methods. One retrieves an instance of load balancer and Cobrowse Server. The other method gets the conavigation channel ID. This information will be used later in the `CoBrowseEventHandler.aspx` file.

Getting Instances of Load Balancer and Cobrowse Server

The initial try-catch script block of the `CoBrowseEventHandler.aspx` file creates an instance of load balancer and attempts to discover a Cobrowse Server host:

```
protected string GetCobrowseHost()
{
    SimpleSamplesConstants ssc = new SimpleSamplesConstants();
    string CoBrowseServerHost = "";
    try
    {
        ServiceInfo si = LoadBalancer.GetServiceInfo(CfgAppType.CFGCoBrowsingServer,
            ssc.TenantName);
        if (si != null)
            CoBrowseServerHost = si.Host;
        else
            CoBrowseServerHost = ssc.ConavServerUrl;
    }
    catch (Exception ex)
    {
        CoBrowseServerHost = ssc.ConavServerUrl;
    }
    return CoBrowseServerHost;
}
```

Getting the Conavigation Channel ID

The retrieves the `SimpleSamplesConstants.ConavChannelID` to be later used in the `CoBrowseEventHandler.aspx` file:

```
protected string GetConavigationiChannelID()
{
    SimpleSamplesConstants ssc = new SimpleSamplesConstants();
    return ssc.ConavChannelID;
}
```

Drawing the Form

This HTML code block draws a page and defines links to perform basic cobrowse functions:

```
<body onload="javascript:window_onload();" onunload= "javascript:window_onunload ();"
    class="samplesBody">
<script language="javascript" src="../../CommLib.js" type="text/javascript"></script>

<script language="javascript" type="text/javascript" >
    var CobrowseHostName = "<%=GetCobrowseHost()%>";
    var ConavigationiChannelID = "<%=GetConavigationiChannelID()%>";
</script>
```

```

<form name="InfoForm" id="InfoForm" onsubmit="return false;"
action="CoBrowseEventHandler.aspx">
  <table style="width:100%" cellpadding="2">
    <tr>
      <td class="samplesSampleTopToolbar">
        <table>
          <tr>
            <td style="width:435px"
class="samplesSampleNameTitleWhite">Basic Cobrowse Sample</td>
            <td style="width:91px" onclick="javascript:CoBrowse_onclick();"
class="sampleGreenButtonBig" onmouseover="javascript:MouseOver(this);"
onmouseout="javascript:MouseOut(this);">Start</td>
            <td style="width:91px"
onclick="javascript:EndCoBrowse_onclick();" class="sampleRedButtonBig"
onmouseover="javascript:MouseOver(this);"
onmouseout="javascript:MouseOut(this);">Stop</td>
            <td style="width:auto" ></td>
          </tr>
        </table>
      </td>
    </tr>
    <tr>
      <td class="samplesSampleParagraph">Basic cobrowse example. Demonstrates how
to cobrowse with an agent.</td>
    </tr>
    <tr>
      <td height="100%">
        <table style="width:617px" border="0">
          <tr>
            <td class="samplesSampleFieldTitle" ></td>
            <td></td>
            <td style="width:6px; vertical-align: middle;"></td>
          </tr>
          <tr>
            <td class="samplesSampleFieldTitle" >Events:</td>
            <td><textarea style="width:517px; height:200px"
class="samplesFieldStyle" rows="10" cols="10" id="Messages" name="Messages"
readonly="readonly"></textarea></td>
            <td style="width:6px; vertical-align: middle;"></td>
          </tr>...
        </table>
      </td>
    </tr>
  </table>

```

Declarations

This block of JavaScript then declares and initializes core variables and functions. The `window_onunload()` function is declared, but it is not implemented:

```
<script language="javascript" type="text/javascript">
var HBApiWindow = null;
var HBUserID = "";
var IsFirstConavigation = true;
var IsAgentJoined = false;

var UserLoggedIn= false;
var ConnectTo = "";
var StartPage = "http://www.google.com";

function window_onload()
{
    AddMessage ("Cobrowse server host name: " + CobrowseHostName);
}
function window_onunload () {}
```

The `AddMessage()` function is declared here, and it is used throughout “Event Handlers” on [page 313](#) to display informational and diagnostic messages.

```
function AddMessage (str)
{
    document.forms[0].Messages.value = document.forms[0].Messages.value + "\r\n" + str;
    document.forms[0].Messages.scrollTop = document.forms[0].Messages.scrollHeight;
}
```

The next block of code checks for redundant logins, and then it calls two functions: `HBInitializeAPI()` for initializing the Cobrowse API, and `HBLoginGuest()` for logging in the user properly. These functions are defined in the `hbmessaging.js` file. For details, refer to the KANA Response Live documentation that is listed in “Related Documentation Resources” on [page 363](#).

```
//----- ALL about Cobrowse -----
function CoBrowse_onclick()
{
    if(UserLoggedIn == true)
    {
        alert("Please log out user "+HBUserID+" before starting a new co-browse session.");
    }
    else
    {
        AddMessage("Connecting to Cobrowse server...");...
```

Note the calls to `AddMessage()` above, as well as in the following calls to the external `HBLogout()` function. `HBLogout()` is another function that is defined in the `hbmessaging.js` file and described in the KANA Response Live documentation.

```
function EndCoBrowse_onclick()
{
    if (HBApiWindow != null)
        HBApiWindow.HBLogout();
}

function doExitSession()
{
    AddMessage("Logging out...");
    HBApiWindow.HBLogout();
}
```

Event Handlers

This final section of code defines event handlers for cobrowse requests. Each of these event-handler functions typically calls a function of the same name in the `hbmessaging.js` file. For details about those external functions, refer to the KANA Response Live documentation that is listed in “Related Documentation Resources” on [page 363](#).

Each of these internal functions also calls the `AddMessage()` function (defined earlier in this file) to display informational or diagnostic messages.

```
/***** HB API Events handlers *****/
function HBCouldNotConnect( sReasonID )
{
    AddMessage("Can't connect to Cobrowse server. Reason: "+sReasonID);
}

function HBLoginError(reasonID, description)
{
    AddMessage("Can't login to Cobrowse server. Reason: " + reasonID + " Description: " +
        description);
}

function HBJoinedSuccessfully()
{
    IsAgentJoined = true;
    AddMessage ("User joined.");
}

function HBJoinRequested(sName)
{
    AddMessage ("Event HBJoinRequested(" + sName + ")");
    return true;
}
```

```

function HBSessionEnded()
{
    AddMessage("Conaviagation session has ended.");
}

function HBLoggedIn(sHipboneID)
{
    HBUserID = sHipboneID;
    AddMessage ("CobrowseID : " + sHipboneID + ".");
    UserLoggedIn = true;
    HBApiWindow.HBCreateSession();
}

function HBLoggedOut(sReasonID)
{
    AddMessage("You have been logged out.");
    UserLoggedIn = false;
    HBUserID = "";
}

function HBSessionStarted()
{
    AddMessage ("HBSessionStarted()");
    HBApiWindow.HBConavigateLink(StartPage, null);
}

function HBLinkConavigated(sLink, sTarget, sUserName)
{
    AddMessage ("Event HBLinkConavigated: " + sLink);
}

function HBUserExitedSession(name)
{
    AddMessage ("Event HBUserExitedSession(" + name + ")");
    if (HBUserID == name)
        doExitSession();
}

function HBUserEnteredSession(name)
{
    AddMessage ("User with ID : " + name + " has joined to session.");
    //HBApiWindow.HBReload("HB_HIPBONE"); // fix for join bug
}...

```

Chat and Cobrowse Sample

The .../ChatAndCoBroswe directory contains the files for the Chat and Cobrowse Sample.

Purpose

The Chat and Cobrowse Sample demonstrates how to use cobrowsing functionality during a chat session.

Functionality Overview

The following sections outline the code that used for implementing the Chat and Cobrowse Sample:

- “Getting Instances of Load Balancer and CoBrowse Server” on [page 316](#)
- “Functions” on [page 316](#)
- “Event Handlers” on [page 318](#)
- “Drawing the Chat Form and Cobrowse Links” on [page 320](#)

Files

The .../ChatAndCoBroswe directory consists of 13 files. Seven of these are common to all of the Cobrowse Samples and are described in “Cobrowse Samples Overview” on [page 308](#):

- blank.html
- hbapi.html
- hbmessage_to_var.html
- hbmessagingform.html
- hbmessage_to_var.js
- hbmessaging.js
- qstring.js

Four files are shared with the Chat Sample and are detailed in “Chat Sample” on [page 238](#):

- ChatCommand.aspx and ChatCommand.aspx.cs—Virtually identical to its Chat Sample counterpart.
- ChatPanel.aspx and ChatPanel.aspx.cs—Contains added code (compared to its Chat Sample counterpart) that provides cobrowsing functionality. See the detailed code explanation in the next section.

The last following two files will be explained in detail here:

- ChatTranscript.aspx—Virtually identical to its Advance Chat Sample counterpart.
- ChatAndCoBrowse.htm—The main frameset of this sample.

Code Explanation

The following subsections explain the code that is in the Chat and Cobrowse Sample.

Getting Instances of Load Balancer and CoBrowse Server

The following try/catch block is found in the ChatCommand.aspx.cs file of the Chat and Cobrowse Sample. It creates a load balancer instance and attempts to discover a Cobrowse Server host:

```
try
{
    bServiceAvailable = false;
    ServiceInfo si = LoadBalancer.GetServiceInfo(CfgAppType.CFGChatServer,
        ssc.TenantName);
    svcHost          = si.Host;
    svcPort          = si.WebApiPort;
    str_chat_alias   = si.Alias;
    bServiceAvailable = true;
}
catch (LoadBalancerException e1)
{
    str_itf_response = "ERROR";
    str_itf_message = "Chat service is not available at this time. Please try it later.";
}
```

Functions

As in the ChatPanel.aspx file of the Chat Sample, this file includes a show_message() function. Elsewhere in this file, cobrowsing event handlers will call the show_message() function to write messages to the chat frame:

```
function show_message(strNickName, strMessage, iUserType)
{
    //iUserType == 0 from Agent
    //iUserType == 1 from Client
    //iUserType == 2 from External
    //iUserType == 3 from Supervisor
    //iUserType == 4 from System
    //iUserType == 5 command from System to push URL from agent
    if (iUserType == 0)
    {
        window.frames.ChatTranscript.AddMessage(strNickName + ":", AgentNickNameColor, 1,
```

```

        1);
        window.frames.ChatTranscript.AddMessage(strMessage, AgentMessageColor, 1, 0);
    }
    else if (iUserType == 1)
    {
        window.frames.ChatTranscript.AddMessage(strNickName + ":", ClientNickNameColor, 1,
            1);
        window.frames.ChatTranscript.AddMessage(strMessage, ClientMessageColor, 1, 0);
    }
    else if (iUserType == 2 || iUserType == 3)
    {
        window.frames.ChatTranscript.AddMessage(strNickName + ":", AgentNickNameColor, 1,
            1);
        window.frames.ChatTranscript.AddMessage(strMessage, AgentMessageColor, 1, 0);
    }
    else if (iUserType == 4)
    {
        if (strMessage == "Agent is typing.")
        {
            show_agent_typing(strNickName);
        }
        else if (strMessage == "Agent has stopped typing.")
        {
            hide_agent_typing();
        }
        else
        {
            window.frames.ChatTranscript.AddMessage("System:", ActionColor, 1, 1);
            window.frames.ChatTranscript.AddMessage(strMessage, ActionColor, 1, 0);
        }
    }
    else if (iUserType == 5)
    {
        hide_agent_typing();
        var bNeedToShowMessage = false;
        var winNewWindow = window.open(strMessage, "PushUrLWindow" + iOpenWindowCounter,
            "", false);
        iOpenWindowCounter++;

        if (winNewWindow == null)
            bNeedToShowMessage = true;
        else
        {
            if (window.opera)
            {
                if (!winNewWindow.opera)
                    bNeedToShowMessage = true;
            }
        }
    }

    if (bNeedToShowMessage)

```

```

{
    window.frames.ChatTranscript.AddMessage("System: pop-up blocker detected. ",
        ActionColor, 1, 1);
    window.frames.ChatTranscript.AddMessage("Please navigate to the next link: " +
        strMessage, ActionColor, 1, 0);...

```

As in the `CoBrowseEventHandler.aspx` file of the Cobrowse Sample, this version of `ChatPanel.aspx` includes additional logic, functions, and event handlers to support interactions with the Cobrowsing Server.

```

function CoBrowse_onclick()
{
    if(UserLoggedIn == true)
    {
        alert("Please log out user "+HBUserID+" before starting a new co-browse
            session.");
    }
    else
    {
        show_system_message("Connecting to Cobrowse server " + CobrowseHostName + " ...");
        if (HBApiWindow == null)
        {
            HBApiWindow = parent.hbapi;
            var strUrl = new String(window.location.href);
            var strProcessorUrl= strUrl.substring(0,
                strUrl.lastIndexOf("/")+"/"+hbmessage_to_var.html";
            HBApiWindow.HBInitializeAPI("ChatFrame", CobrowseHostName, strProcessorUrl);
            HBApiWindow.HBLoginGuest(ConavigationiChannelID, "", ConnectTo, false,
                "acctSpecificData");
        }
        else
        {
            HBApiWindow.HBLoginGuest(ConavigationiChannelID, "", ConnectTo, false);
        }...
    }
}

```

Event Handlers

The first three event handlers are similar to identically named event handlers in the `CoBrowseEventHandler.aspx` file of the Cobrowse Sample. However, whereas the event handlers of that file call an `AddMessage()` function, the event handlers of this file instead call its `show_message()` function (see “Functions” on [page 316](#)), so as to write their messages to the chat frame. Note that these messages are not sent to the agent and are not visible in a chat transcript:

```

/**** HB API Events handlers ****/
function HBCouldNotConnect( sReasonID )
{
    show_system_message("Can't connect to Cobrowse server. Reason: "+sReasonID);
}

function HBLoginError(reasonID, description)

```

```

{
    show_system_message("Can't login to Cobrowse server. Reason: " + reasonID + "
        Description: " + description);
}

function HBJoinedSuccessfully()
{
    IsAgentJoined = true;
    show_system_message ("Agent joined.");
}

function HBSendCobrowseID()
{
    clearTimeout(timerID);
    str_cmd = "send";
    str_msg2send = "$$CUSTOMER_COBROWSING_ID=" + HBUserID;
    httpRequest("POST", "ChatCommand.aspx", true, handleResponse, prepareQueryString());
}

function HBLoggedOut(sReasonID)
{
    show_system_message("You have been logged out.");
    UserLoggedIn = false;
    HBUserID = "";
}

function HBSessionStarted()
{
    show_system_message ("HBSessionStarted()");
    HBApiWindow.HBConavigateLink(StartPage, null);
}

function HBLinkConavigated(sLink, sTarget, sUserName)
{
    show_system_message ("Event HBLinkConavigated: " + sLink);
}

function HBUserExitedSession(name)
{
    show_system_message ("Event HBUserExitedSession(" + name + ")");
    if (HBUserID == name)
        doExitSession();
}

function HBUserEnteredSession(name)
{
    show_system_message ("User " + name + " has entered session.");
}

```

The HBJoinRequested() and the HBSessionEnded() event handlers of this file differs in the same way from its counterpart in the Cobrowse Sample. For

example, instead of calling the `AddMessage()` function it calls the `show_message()` function to identify the agent to the caller:

```
function HBJoinRequested(sName)
{
    show_system_message ("Your agent is : " + sName + ".");
    return true;
}
```

```
function HBSessionEnded()
{
    show_system_message("Conaviagation session has ended.");
}
```

The `HBLoggedIn()` event handler of this file differs in two ways from its counterpart in the Cobrowse Sample. First, it calls the `show_message()` function instead of the `AddMessage()` function. Second, upon verifying the login and connection of the customer, it writes the Cobrowse ID of the customer to the chat frame before it calls the `HBCreateSession()` function:

```
function HBLoggedIn(sHipboneID)
{
    HBUserID = sHipboneID;
    show_system_message ("CobrowseID : " + sHipboneID + ".");
    UserLoggedIn = true; ...
```

Specifically, this next code block sends the encoded message to the agent via chat. This way, the agent will know where to connect to begin cobrowsing with the customer:

```
...if (bConnected == true)
{
    HBSendCobrowseID();
}
HBApiWindow.HBCreateSession();
}
```

Finally, each of the remaining event handlers of this file differs from its counterpart of the Cobrowse Sample by internally calling the `show_message()` function instead of the `AddMessage()` function:

- `HBLoggedOut()`
- `HBSessionStarted()`
- `HBLinkConavigated()`
- `HBUserExitedSession()`
- `HBUserEnteredSession()`

Drawing the Chat Form and Cobrowse Links

Along with the HTML code that creates the traditional chat form, the code displays the host name of the Cobrowsing Server, and displays links for

starting and stopping a cobrowse session. It also provides check boxes for both a transcript request and a survey request:

```
<form action="ChatPanel.aspx" id="chat_form" onsubmit="javascript:on_send(); return
false;">
  <table style="width:100%" cellpadding="2">
    <tr>
      <td class="samplesSampleTopToolbar">
        <table>
          <tr>
            <td style="width:253px" class="samplesSampleNameTitleWhite">Chat
and Cobrowse Sample</td>
            <td style="width:91px;font:8pt verdana;"
onclick="javascript:on_connect();" class="sampleGreenButtonBig"
onmouseover="javascript:MouseOver(this);"
onmouseout="javascript:MouseOut(this);">Start chat</td>
            <td style="width:91px;font:8pt verdana;"
onclick="javascript:on_disconnect();" class="sampleRedButtonBig"
onmouseover="javascript:MouseOver(this);"
onmouseout="javascript:MouseOut(this);">Stop chat</td>
            <td style="width:91px;font:8pt verdana;"
onclick="javascript:CoBrowse_onclick();" class="sampleGreenButtonBig"
onmouseover="javascript:MouseOver(this);"
onmouseout="javascript:MouseOut(this);">Start cobrowse</td>
            <td style="width:91px;font:8pt verdana;"
onclick="javascript:EndCoBrowse_onclick();" class="sampleRedButtonBig"
onmouseover="javascript:MouseOver(this);"
onmouseout="javascript:MouseOut(this);">Stop cobrowse</td>
            <td style="width:auto" ></td>
          </tr>
        </table>
      </td>
    </tr>
    <tr>
      <td class="samplesSampleParagraph">Fill in your personal information in
fields below to proceed.</td>
    </tr>
    <tr>
      <td>
        <table style="width:617px" border="0">
          <tr>
            <td class="samplesSampleFieldTitle" ></td>
            <td></td>
            <td style="width:6px; vertical-align: middle;"></td>
          </tr>
          <tr>
            <td class="samplesSampleFieldTitle" >First name:</td>
```

[illegible]

Cobrowse with Meet Me

The `.../CoBrosweMeetMe` directory contains the files for the Cobrowse with Meet Me Sample. This sample demonstrates cobrowse-with-meet-me functionality.

Purpose

The Cobrowse with Meet Me Sample code demonstrates how to set up a cobrowse session with a specific other person whose cobrowse ID the user knows.

Functionality Overview

The following sections outline the code used to implement the Cobrowse with Meet Me Sample:

- “Entering the Cobrowse ID of the Other Party” on [page 323](#)
- ““Meeting” the Other Party” on [page 324](#)
- “Event Handlers” on [page 325](#)

Files

The .../CoBrosweMeetMe directory contains nine files; eight of these are documented in “Cobrowse Samples Overview” on [page 308](#). This section focuses on code that is unique to CoBrowseEventHandler.aspx file of this sample (as compared to the corresponding file in “Basic Cobrowse Sample” on [page 309](#)).

Code Explanation

The following subsections explain the code that is in the Cobrowse with Meet Me Sample.

Entering the Cobrowse ID of the Other Party

Compared to the CoBrowseEventHandler.aspx file of the Cobrowse Sample, this sample adds an input box and a link by which the user can specify the cobrowse ID of the person with whom the user wants to browse with:

```
<form name="InfoForm" id="InfoForm" onsubmit="return false;"
  action="CoBrowseEventHandler.aspx">
  <table style="width:100%" cellpadding="0" cellspacing="0" border="0">
    <tr>
      <td class="samplesSampleTopToolbar">
        <table>
          <tr>
            <td style="width:507px"
class="samplesSampleNameTitleWhite">Cobrowse &quot;Meet Me&quot; Sample</td>
            <td style="width:91px" onclick="javascript:CoBrowse_onClick();"
class="sampleGreenButtonBig" onmouseover="javascript:MouseOver(this);"
onmouseout="javascript:MouseOut(this);">Start</td>
```

```

        <td style="width:91px"
onclick="javascript:EndCoBrowse_onclick();" class="sampleRedButtonBig"
onmouseover="javascript:MouseOver(this); "
onmouseout="javascript:MouseOut(this); ">Stop</td>
        <td style="width:auto" ></td>
    </tr>
</table>
</td>
</tr>
<tr>
    <td class="samplesSampleParagraph">Cobrowse functionality sample. Also
allows to connect to another user by known Cobrowse ID of this user.</td>
</tr>

<tr>
    <td>
        <table border="0">
            <tr>
                <td class="samplesSampleFieldTitle" ></td>
                <td></td>
                <td style="width:6px; vertical-align: middle;"></td>
            </tr>
            <tr>
                <td class="samplesSampleFieldTitle" >Connect to:</td>
                <td><input class="samplesFieldStyle" type="text" id="ConnectTo" value=""
name="ConnectTo"/></td>
                <td style="width:6px; vertical-align: middle;"><font
class="samplesMandatoryMark">*</font></td>
            </tr>...

```

“Meeting” the Other Party

The preceding code captures a ConnectTo value, which represents the target cobrowse ID. In the following code, the CoBrowse_onclick() function of this sample includes additional logic that attempts to connect to this ID and notifies the user if the ConnectTo value is empty:

```

function CoBrowse_onclick()
{
    ConnectTo = trim(document.forms[0].ConnectTo.value);

    if(UserLoggedIn == true)
    {
        alert("Please log out user "+HBUserID+" before starting a new co-browse
        session.");
    }
    else if(ConnectTo == "")
    {

```

```

    alert("Please enter customer id you're trying to connect to");
}
else
{
    AddMessage("Connecting to Cobrowse server " + CobrowseHostName + " ...");
    if (HBApiWindow == null)
    {
        HBApiWindow = parent.hbapi;
        var strUrl = new String(window.location.href);
        var strProcessorUrl = strUrl.substring(0,
            strUrl.lastIndexOf("/")+"/hbmessage_to_var.html");
    }
}

```

Notice in the following code, that the `ConnectTo` variable is passed into both the `HBApiWindow.HBInitializeAPI` method and the `HBApiWindow.HBLoginGuest` method.

```

HBApiWindow.HBInitializeAPI("EventHandlerFrame", CobrowseHostName,
    strProcessorUrl);
HBApiWindow.HBLoginGuest(ConavigationiChannelID, "", ConnectTo, false,
    "acctSpecificData");
}
else
{
    HBApiWindow.HBLoginGuest(ConavigationiChannelID, "", ConnectTo, false);
}...

```

Event Handlers

Most of the event handlers of this file correspond to those that are in the `CoBrowseEventHandler.aspx` file of the Basic Cobrowse Sample. However, there are minor differences.

The `HBLoggedIn()` event handler of this file omits the internal call to `HBApiWindow.HBCreateSession()` that is found in the `HBLoggedIn()` event handler of the Basic Cobrowse Sample because you can assume that the other party has already created the session:

```

function HBLoggedIn(sHipboneID)
{
    HBUserID = sHipboneID;
    AddMessage ("CobrowseID : " + sHipboneID + ".");
    UserLoggedIn = true;
}

```

Similarly, the `HBSessionStarted()` event handler of this file differs from its counterpart in the Cobrowse Sample by omitting an internal call to `HBApiWindow.HBConavigateLink(StartPage, null)`. Again, you can assume that the other party has already established the URL to cobrowse:

```

function HBSessionStarted()
{

```

```

        AddMessage ("HBSessionStarted()");
    }

```

For similar reasons, this file's `HBUserEnteredSession()` event handler omits its Cobrowse Sample counterpart's internal call to `HBApiWindow.HBConavigateLink(StartupPage, null)`:

```

function HBUserEnteredSession(name)
{
    AddMessage ("User with ID : " + name + " has joined to
        session.");
    //HBApiWindow.HBReload("HB_HIPBONE"); // fix for join bug
}

```

Cobrowse with Initial Startup Page

The `.../CoBrowseInitStartupPage` directory contains the files for the Cobrowse with Initial Startup Page Sample.

Purpose

The code for the Cobrowse with Initial Startup Page Sample demonstrates basic cobrowsing that uses a specified initial startup page.

Functionality Overview

The following sections outline the code that is used to implement the Cobrowse with Initial Startup Page Sample:

- “Specifying the Startup Page” on [page 327](#)
- “Event Handlers” on [page 328](#)

Files

This section focuses on code that is unique to this sample, as compared to the basic “Basic Cobrowse Sample” on [page 309](#). The `.../CoBrowseInitStartupPage` directory contains ten files; eight of these are common files that already have been described in “Cobrowse Samples Overview” on [page 308](#). Two files differentiate this sample:

- `InitialStartupPageExample.html`—Replaces the `CoBrowse.htm` file of the Basic Cobrowse Sample, but serves a similar function: Sets up the basic display frame as a container for imported logic.
- `CoBrowseEventHandler.aspx` and `CoBrowseEventHandler.aspx.ca`—Contain additional logic, compared to the corresponding Cobrowse Sample file of the same name. For details, see the following section.

Code Explanation

The following subsections explain the code that is in the Cobrowse with Initial Startup Page Sample.

Specifying the Startup Page

Compared to its counterpart in the Basic Cobrowse Sample, the `CoBrowseEventHandler.aspx` file of this sample contains one additional code block that provides:

- An input box to specify the cobrowse ID of the other party.
- An input box to confirm or override the initial URL that both users will cobrowse.
- A link to open that page.

The following shows the added code:

```
<td class="samplesSampleTopToolbar">
    <table >
        <tr >
            <td style="width:511px"
class="samplesSampleNameTitleWhite">Cobrowse Init Start Page Sample</td>
                <td style="width:91px" onclick="javascript:CoBrowse_onClick();"
class="sampleGreenButtonBig" onmouseover="javascript:MouseOver(this);"
onmouseout="javascript:MouseOut(this);">Start</td>
                <td style="width:91px"
onclick="javascript:EndCoBrowse_onClick();" class="sampleRedButtonBig"
onmouseover="javascript:MouseOver(this);"
onmouseout="javascript:MouseOut(this);">Stop</td>
                <td style="width:auto" ></td>
            </tr>
        </table>
    </td>
</tr>
<tr>
    <td class="samplesSampleParagraph">Cobrowse functionality sample which also
allows start session from particular page.</td>
</tr>

<tr>
    <td>
        <table border="0">
            <tr>
                <td class="samplesSampleFieldTitle" ></td>
                <td></td>
                <td style="width:6px; vertical-align: middle;"></td>
            </tr>
```

```

        <tr>
            <td class="samplesSampleFieldTitle" >Connect to:</td>
            <td><input class="samplesFieldStyle" type="text" id="ConnectTo" value=""
name="ConnectTo"/></td>
            <td style="width:6px; vertical-align: middle;">&nbsp;</td>
        </tr>
        <tr>
            <td class="samplesSampleFieldTitle" >Initial startup page:</td>
            <td><input class="samplesFieldStyle" type="text" id="StartPage"
value="http://www.yahoo.com" name="StartPage"/></td>
            <td style="width:6px; vertical-align: middle;"><font
class="samplesMandatoryMark">*</font></td>
        </tr>

        <tr>
            <td class="samplesSampleFieldTitle" >Events:</td>
            <td><textarea style="height:200px" class="samplesFieldStyle" rows="10"
cols="60" id="Messages" name="Messages" readonly="readonly"></textarea></td>
            <td style="width:6px; vertical-align: middle;"></td>
        </tr>...

```

Event Handlers

Most of the event handlers of this file correspond to those event handlers in the `CoBrowseEventHandler.aspx` file of the Basic Cobrowse Sample. This section identifies the minor differences.

In the version of the `HBLoggedIn()` event handler of this file, the internal call to `HBApiWindow.HBCreateSession()` is embedded in an `if` branch. This `if` branch verifies a connection to the other party before it creates a session:

```

function CoBrowse_onclick()
{
    if(UserLoggedIn == true)
    {
        alert("Please log out user "+HBUserID+" before starting a new cobrowse session.");
    }
    else
    {
        ConnectTo = trim(document.forms[0].ConnectTo.value);
        StartPage = trim(document.forms[0].StartPage.value);
        if(StartPage == "")
        {
            alert ("Please specify initial startup page.");
            return;
        }
        AddMessage("Connecting to Cobrowse server " + CobrowseHostName + " ...");

        if (HBApiWindow == null)

```

```

{
    HBApiWindow = parent.hbapi;
    var strUrl = new String(window.location.href);
    var strProcessorUrl = strUrl.substring(0,
        strUrl.LastIndexOf("/")+"/hbmessage_to_var.html");

    HBApiWindow.HBInitializeAPI("EventHandlerFrame", CobrowseHostName, strProcessorUrl);
    HBApiWindow.HBLoginGuest(ConavigationiChannelID, "", ConnectTo, false,
        "acctSpecificData");
}
else
{
    HBApiWindow.HBLoginGuest(ConavigationiChannelID, "", ConnectTo, false);...}...

```

The HBSessionStarted() event handler of this file differs from its Basic Cobrowse Sample counterpart by omitting an internal call to HBApiWindow.HBConavigateLink(StartPage, null). You can assume that the other party has already established both a cobrowse session and the URL to cobrowse:

```

function HBSessionStarted()
{
    AddMessage ("HBSessionStarted()");
}

```

The HBApiWindow.HBConavigateLink(StartPage, null) call instead occurs in the HBUserEnteredSession() event handler of this file. (The Basic Cobrowse Sample version of that event handler makes no such call.) The call is embedded in an if branch that first verifies the Cobrowse ID of the other party. It conavigates to a start page only if the party has newly entered the session:

```

function HBUserEnteredSession(name)
{
    AddMessage ("User with ID : " + name + " has joined to
        session.");
    //HBApiWindow.HBReload("HB_HIPBONE"); // fix for join bug
    if(HBUserID == name)
    {
        HBApiWindow.HBConavigateLink(StartPage, null);
    }
}

```

Cobrowse with Dynamic Startup Page Sample

The `.../CoBrosweDynamicStartPage` directory contains the Cobrowse with Dynamic Startup Page Sample.

Purpose

The Cobrowse with Dynamic Startup Page Sample demonstrates how one party can click a cobrowse control on an HTML page that contains a form. When Cobrowse begins, all of the form data that already is entered is dynamically prefilled in the Cobrowse window of the other party, so that a customer and an agent can access the form data of the other dynamically.

Functionality Overview

The following sections outline the code that is used to implement the Cobrowse with Dynamic Startup Page Sample:

- “Functions” on [page 331](#)
- “Getting User Data” on [page 331](#)

Files

The `.../CoBrosweDynamicStartPage` directory contains five files. In order for the sample to work properly, place all the files together into a public web directory. Four of these files (listed in “Cobrowse with Dynamic Startup Page” on [page 56](#)) should not be modified.

This section focuses on the file that you can modify—the sample’s main file; `ExampleOfDynamicStartPage.aspx`. This file serves some of the same purposes as the `CoBrowseEventHandler.aspx` file in “Basic Cobrowse Sample” on [page 309](#); therefore, this discussion focuses on the unique code that it provides.

The Cobrowse session starts from the `ExampleOfDynamicStartPage.aspx` page, where it collects user input. To check the sample, enter arbitrary information in the form fields of this page, then click the `Live Help` link. You will see a cobrowse window that is specific to the dynamic-startup page.

Code Explanation

The following subsections explain the code that is in the Cobrowse with Dynamic Startup Page Sample.

Functions

The sample code defines an `openLiveHelp()` function. This function opens the Dynamic Startup Page API window when the user clicks the Live Help link:

```
function openLiveHelp()
{
    startDSPMeetMe(ConavigationiChannelID, null, "guest", null,
        CobrowseHostName)
}
```

Getting User Data

The file then builds the form that solicits the input of personal information by the user. Two text boxes prompt for the name and e-mail address of the user:

```
<tr>
    <td class="samplesSampleFieldTitle" >Full Name:</td>
    <td><input style="width:550px; vertical-align: middle;" type="text"
class="samplesFieldStyle" value="" name="fullName"/></td>
    <td style="width:6px; vertical-align: middle;"><font
class="samplesMandatoryMark">*</font></td>
</tr>

<tr>
    <td class="samplesSampleFieldTitle" >Email:</td>
    <td><input style="width:550px; vertical-align: middle;" type="text"
class="samplesFieldStyle" value="" name="emailAddress"/></td>
    <td style="width:6px; vertical-align: middle;"><font
class="samplesMandatoryMark">*</font></td>
</tr>
```

A very basic drop-down list offers three State/Province options; then another text box prompts for the password of the user:

```
<tr>
    <td class="samplesSampleFieldTitle" >State</td>
    <td>
        <select style="width:550px; vertical-align: middle;"
class="samplesFieldStyle" name="state" id="state">
            <option value="">State/Province</option>
            <option value="CA">CA</option>
            <option value="NY">NY</option>
            <option value="TX">TX</option>
        </select>
    </td>
```

```

        </td>
        <td style="width:6px; vertical-align: middle;"><font
class="samplesMandatoryMark">*</font></td>
    </tr>

    <tr>
        <td class="samplesSampleFieldTitle" >Password:</td>
        <td><input style="width:550px; vertical-align: middle;"
class="samplesFieldStyle" type="password" value="" name="password"/></td>
        <td style="width:6px; vertical-align: middle;"><font
class="samplesMandatoryMark">*</font></td>
    </tr>

```

Next, a radio-button panel prompts the user to select a credit-card type:

```

<tr>
    <td class="samplesSampleFieldTitle" >Credit card type:</td>
    <td style="width:550px; vertical-align: middle;">
        <table style="width:100%" class="samplesFieldStyle" border="0">
            <tr>
                <td style="max-width:10px; vertical-align: middle;">
                    <input type="radio" value="Visa" checked="checked"
name="creditCardType" id="creditCardType_0" />
                </td>
                <td>
                    Visa
                </td>
                <td style="max-width:10px; vertical-align: middle;">
                    <input type="radio" value="MasterCard" name="creditCardType"
id="creditCardType_1" />
                </td>
                <td>
                    Mastercard
                </td>
                <td style="max-width:10px; vertical-align: middle;">
                    <input type="radio" value="AmericanExpress"
name="creditCardType" id="creditCardType_2" />
                </td>
                <td>...
            </tr>
        </table>
    </td>

```

A multiple-selection box then allows the user to specify one or more preferred callback times:

```

<tr>
    <td class="samplesSampleFieldTitle" >Preferred callback time:</td>
    <td style="width:550px; vertical-align: middle;">
        <select style="width:100%; height:50px; vertical-align: middle;"
class="samplesFieldStyle" multiple="multiple" name="selectbox1" size="3">
            <option value="Morning">Morning</option>
            <option value="Day time">Day time</option>
        </select>
    </td>

```

```

        <option value="Evening">Evening</option>
    </select>
</td>...

```

Finally, the code defines a link to the Live Help functionality, which is provided by the other files in the installed directory of the sample:

```

<tr >
    <td style="width:549px"
class="samplesSampleNameTitleWhite">Cobrowse Dynamic Start Page Example</td>
    <td style="width:91px" ></td>
    <td style="width:91px" onclick="javascript:openLiveHelp();"
class="sampleGreenButtonBig" onmouseover="javascript:MouseOver(this); "
onmouseout="javascript:MouseOut(this);">Live Help</td>
    <td style="width:auto" ></td>
</tr>...

```

To build your own dynamic startup page example, see “Build Your Own Dynamic Startup Page Example” on [page 221](#).



Chapter

11

eServices Test Tools

This chapter describes the eServices test tools, in three sections:

- [Overview, page 335](#)
- [Using the Tools, page 337](#)
- [Troubleshooting Guide, page 341](#)

Overview

This chapter shows how to access the test tools provided with the eServices web sample installation. The installation wizard prompts you to install these tools.

This chapter does not discuss how the customized server-page files are constructed, nor the code involved in creating them.

Java

In the Java implementation, five test tools are available. You access the main index page for these tools through a URL similar to this one:

```
http://<web_server_hostname>:<web_server_port>/  
TestTool802/index.html
```

where <web_server_hostname>:<web_server_port> is the host and port information for the machine on which you installed the eServices test tools. [Figure 33](#) shows the menu page. Notice that the top frame contains a scrollable list of test tools.

Versions information about MCR API Java classes and servlets.	
 LoadBalancing servlet info page	Shows information about LoadBalancing servlet. Also it provides information about all active servers in current configuration.
Self test information about MCR API Java classes and servlets.	
 E-Mail servers self test page	Self test information about E-Mail servers and possible detected misconfigurations.
 Chat servers self test page	Self test information about Chat servers and possible detected misconfigurations.
 Stat server self test page	Self test information about Stat servers and possible detected misconfigurations.
 Universal Contact Server self test page	Self test information about Universal Contact Servers and possible detected misconfigurations.

Figure 33: Test Tools Menu Loaded Successfully

Warning! Genesys recommends that you restrict public access to your host machine's entire `.../TestTool802` or `...\TestTool802` directory. Pages inside this directory reveal details of your internal network infrastructure—such as host names, ports, and tenant names.

.NET

In the .NET implementation, the 8.x release(s) provides a single load-balancing test tool. You can access this tool through a URL similar to this one:

```
http://<web_svr_host>:<web_svr_port>/WebAPIServer802/
TestTool802/LBTest.aspx
```

Alternatively, you can access this tool in the following way:

1. Load the `index.htm` page. This is available in the top-level directory of the .NET samples, with other common files. (See “Shared Files” on [page 228](#).)
2. Then click `Loadbalancing informational page` link on that page.

Either way, [Figure 35](#) shows the resulting test page.

Using the Tools

Use the tools to verify the configuration of your media servers and to see which media servers are available. The test tools verify the configurations for these components:

- Load-Balancing Servlet
- E-mail Server
- Stat Server
- Chat Server

Load-Balancing Servlet Configuration

Java In the Java implementation, you can test the load-balancing servlet's configuration by clicking either of the first two links in the Test Tools menu page's upper frame. Each of these links shows exactly the same information about the load-balancing servlet and all active servers. The first link is called "LoadBalancing servlet info page." The second is called "Combination of info pages above." [Figure 34](#) shows a successful result from clicking on the first link.

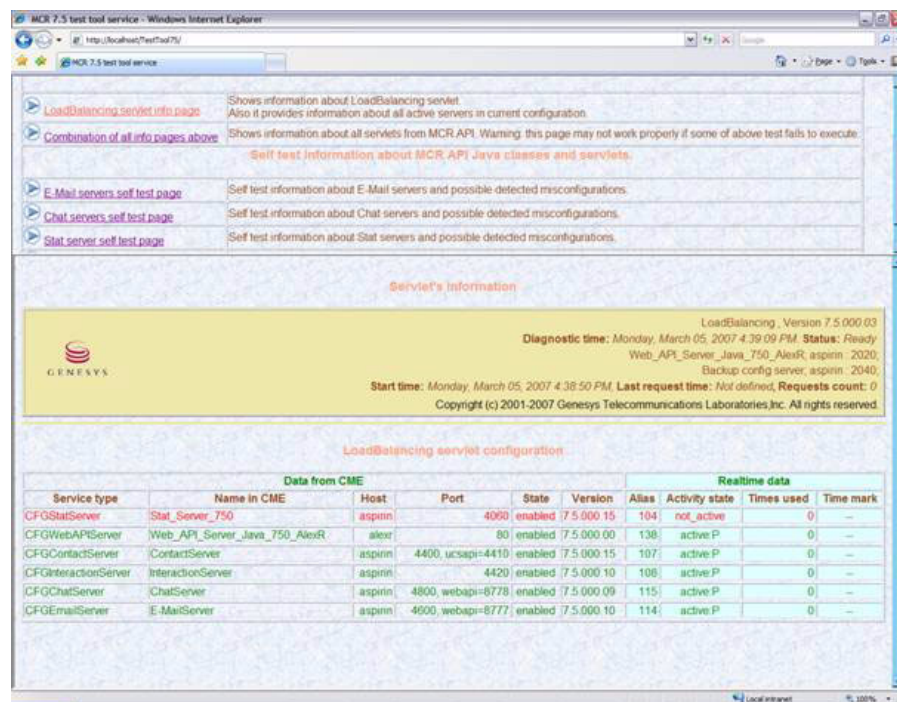


Figure 34: Load-Balancing Test Tool

.NET In the .NET implementation, you can test the load-balancing servlet's configuration by loading the single test tool, as described in “Overview” on page 335. Figure 35 shows a successful load of the resulting test page.

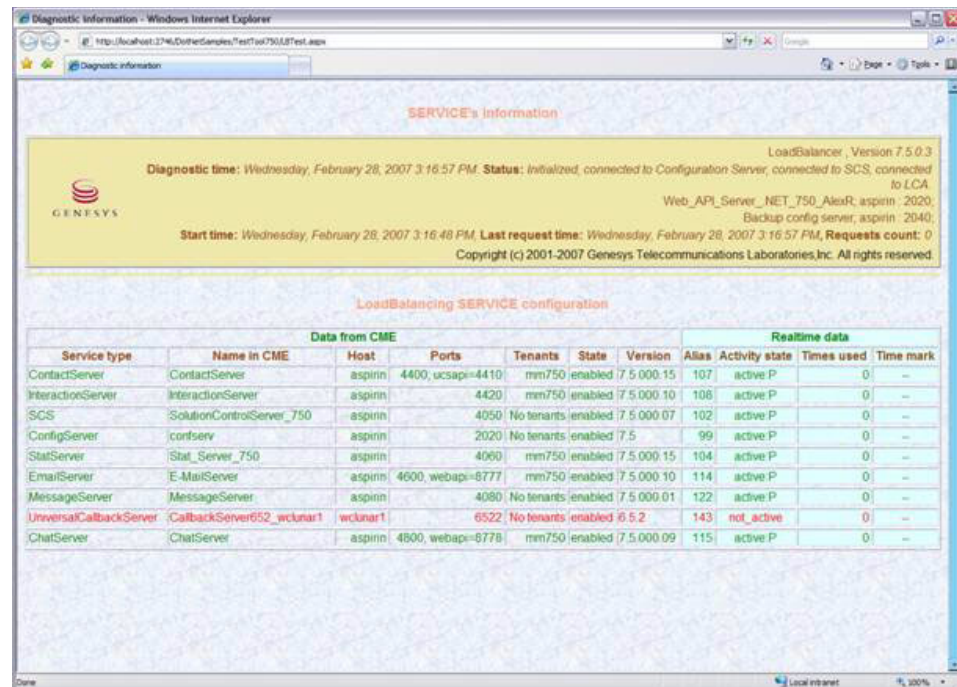


Figure 35: Load-Balancing Test Tool

Verifying Chat Server Configuration

To test the Chat Server configuration, scroll down in the upper frame of the Test Tools menu page and click the “Chat servers self test page” link.

Figure 36 shows a successful test for the Chat Server configuration.

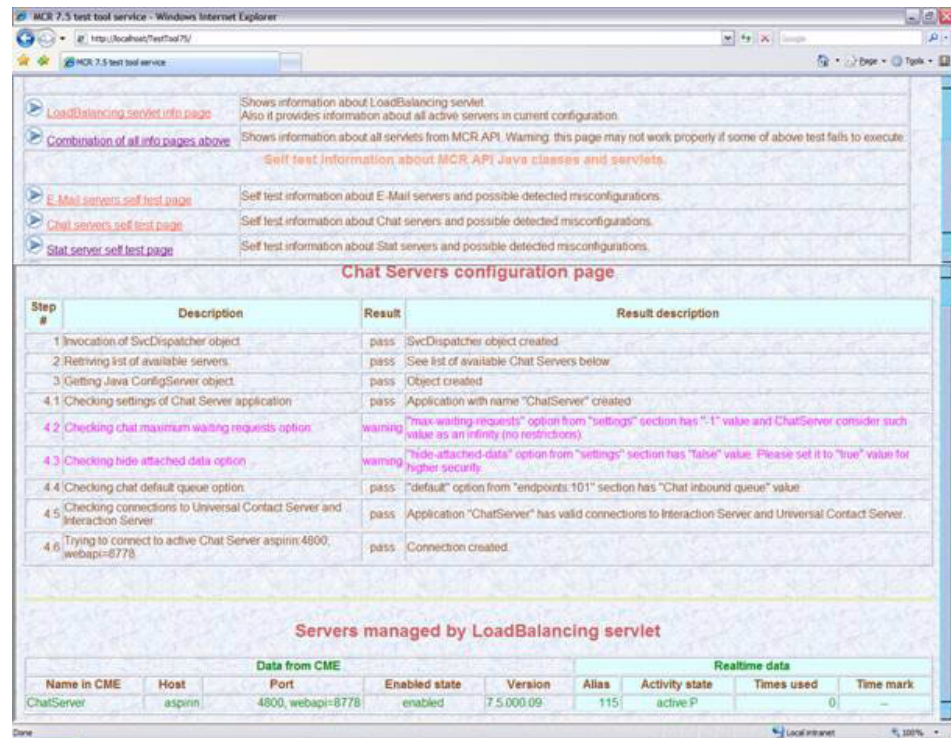


Figure 36: Chat Server Configuration Page

Verifying the Configuration of E-Mail Server

To test the configuration of E-mail Server, scroll down in the upper frame of the Test Tools menu page and click the “E-mail servers self test page” link.

Figure 37 shows a successful test for this configuration.

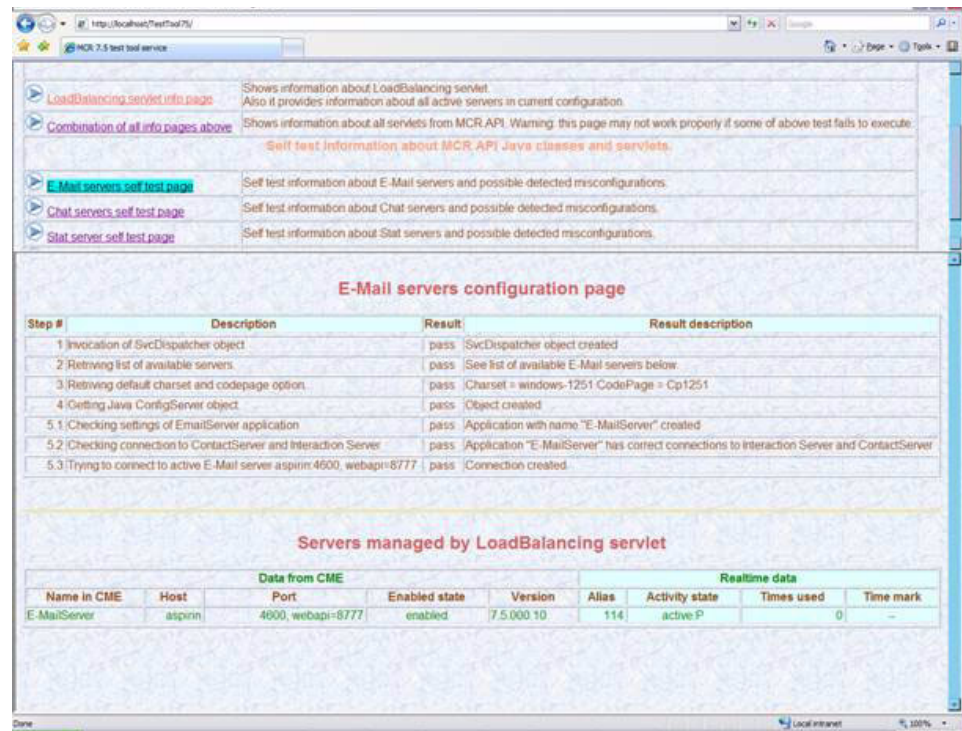


Figure 37: Configuration Page for E-Mail Server

Verifying Stat Server Configuration

To test the Stat Server configuration, scroll down in the upper frame of the Test Tools menu page and click the “Stat server self test page” link. [Figure 38](#) shows a successful test for the Stat Server configuration.

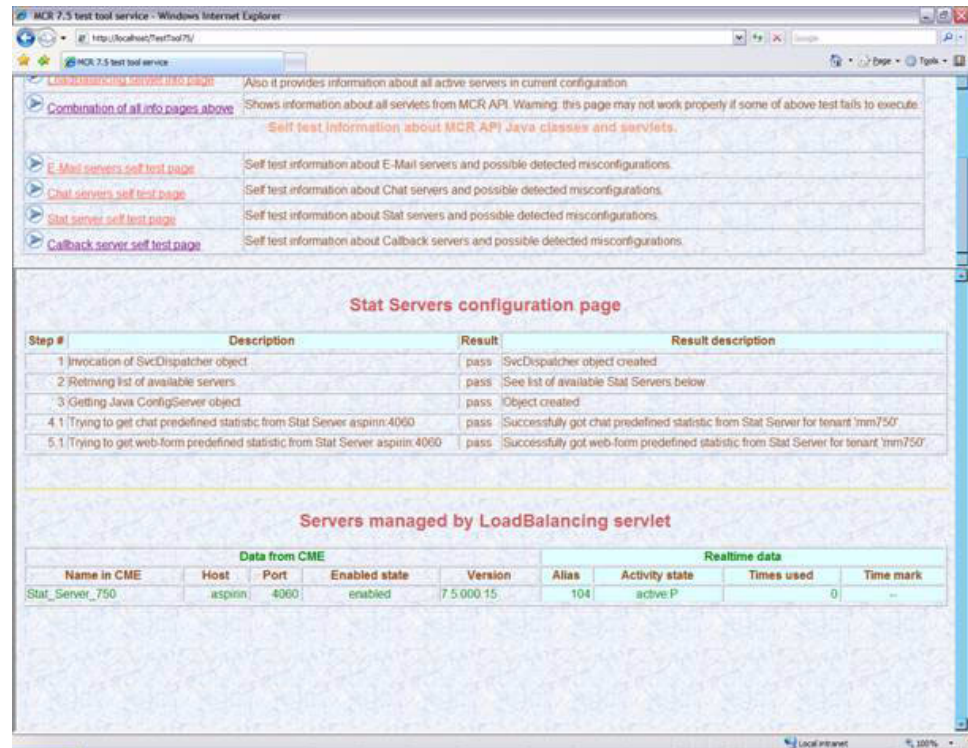


Figure 38: Stat Server Configuration Page

Troubleshooting Guide

The following list provides some suggestions on what to verify if your application is not accessing the media servers properly. However, it is not an exhaustive list. Check the *eServices 8.0 Deployment Guide* for more troubleshooting information.

Data Verification

Check that the following are correct and that there are no spelling errors:

- Media server host name and port
- Web server host name and port
- Servlet engine host name and port
- Application names

Service Availability

Check the following:

- Web server availability
- Servlet engine availability
- Network connection
- Framework availability



Chapter

12 Sample Client Scenarios

This chapter presents some common scenarios in a web client application, and shows how to satisfy them by using code snippets from the web samples. The chapter's goal is to provide you with some ideas for adding media services to your web client application. It discusses the following topics:

- [Disclaimers, page 343](#)
- [Common Scenarios, page 343](#)
- [Conclusion, page 346](#)

Disclaimers

The scenarios listed in this chapter are for illustration only, and are not an exhaustive list. This chapter does not discuss any code outside of the eServices web samples.

If a scenario requires knowledge of general web technology, or of non-Genesys products, the “Potential Solution” section plainly states that requirement. You must research these aspects of the scenario if you are not already familiar with them.

Common Scenarios

Five scenarios are discussed in this section:

- Scenario 1—How to authenticate and assist off-site customers using eServices media services.
- Scenario 2—How to assist website visitors (no user account) using eServices media services.
- Scenario 3—How to assist an on-site customer using eServices media services.

- Scenario 4—How to track user behavior in a web application.
- Scenario 5—How to track user behavior and assist a customer using eServices media services.

Scenario 1

A customer logs into a web application and needs assistance. On the page he is viewing, there is a Help menu and a form. The menu has a list of media options through which a service representative can communicate with the customer. The customer selects one of the options, fills out the related form, and submits it. The contact center receives this form along with the originating page (the page the customer was viewing).

Potential Solution

There are a few tricky points in this scenario. Authentication is one, media availability another.

The authentication scheme in the sample is too rudimentary for a production application. If you have a secure application, you should use HTTPS mode along with user-credential verification. This book does not discuss an authentication scheme. The web offers many tutorials on authentication.

In the Java samples, the `svcDispatcher` class is the one that checks for media availability. The class is more elaborately used in the `IsMediaAvailable()` method in the `CommonScripts.jsp` file.

The `ChatOptions.jsp` code checks for the availability of different chat services. For example, it could block the availability of chat services even when some Chat Servers are available, if a user requests these services during night or weekend hours when no online agent is available.

The e-mail equivalent is in the `Email.jsp` code. The `Common.jsp` file checks for all media.

You can use any common web technology to implement the help function—for example, a drop-down combo box or a menu. Adding the media options to the menu is a bit tricky. It is probably easier for you to examine the code in the customized server-page files just mentioned, and—once you understand that code—incorporate it into your menu code, instead of calling these classes directly.

Scenario 2

A customer with no user account is visiting your website and needs assistance. On the page he or she is viewing, there is a Help menu and an inquiry form. The menu has a list of inquiry topics. The customer selects one of the topics, fills out the related form, and submits it. The contact center receives this form along with the originating page (the page the customer was viewing).

Potential Solution

This is a typical web application scenario. You can use any common web technology to implement the help function—for example, a drop-down combo box or a menu. The form can be an ordinary HTML form along with some JavaScript functions. As for the originating page, you can use a cookie to track which page the user is viewing, or just keep a hidden value somewhere on the form and pass along that value during form submission.

Scenario 3

A customer launches your application at his site, and needs assistance. On the page he is viewing, there is a Help button and a form. The button displays help only through the media option available at that moment. (Your application determines that media option based on contact-center information; no choices are available to the customer.) The customer fills out the related form and submits it. The contact center receives this form, along with the originating page (the page the customer was viewing).

Potential Solution

This scenario is very similar to “Scenario 1” on [page 344](#)—minus the authentication, because the user is on site. You can either follow the suggestion for Scenario 1, minus the authentication section; or expand the authentication scheme to check whether a user is accessing the application from the company network. If the latter is true, your application does not present the login page. Depending on how your network is set up, a simple IP test may be all you need to determine whether the user is on your network.

Scenario 4

A customer is on a specific web application page, and behaves according to a specific pattern. He triggers a flag, and a Help dialog box opens with a request-for-assistance form. The customer fills out the required fields presented in the dialog box, and clicks the Submit button. The contact center receives the request-for-assistance form.

Potential Solution

The implementation for this kind of scenario can quickly get complicated. Even so, you can construct the needed web application without hiring a specialist. You can reconstruct a user behavioral pattern from user events. A pattern can be just a combination of specific links clicked, or of options (radio buttons and check boxes) selected. If the pattern matches your application’s predefined criteria, it triggers a predefined response from your application, such as launching a dialog box.

You can track these user behaviors either by using cookies, or by storing hidden form values, depending on your need. To track form submission or refreshment, use state tracking. Each of the web sample discussions in [Chapter 9](#) has a “Tracking States” subsection that demonstrates how the corresponding sample tracks the form states.

Scenario 5

A customer is on a specific web application page, and triggers a flag that you have created to indicate a customer who may need help without knowing it. Without changing the customer’s view, a hidden criterion determines the media options that might be available to help the customer. If one is available, a Help dialog box appears. If a preferred media type is not available, the dialog box either does not appear, or defaults to any available media.

Potential Solution

This scenario is basically a combination of Scenario 3 and Scenario 4. Use the suggestion for Scenario 4, and add the extra check for media availability from Scenario 3.

Conclusion

This chapter reviewed some common usage scenarios in a web application, and demonstrated how the eServices services can improve the overall user experience.



Appendix

A

Updating Your Customized Code

This chapter compares older versions of eServices's Samples - Java code with the newer 8.0 versions. This chapter covers the following topics:

- [Overview, page 347](#)
- [Common Code, page 347](#)
- [Chat Sample, page 348](#)
- [E-Mail Sample, page 352](#)
- [Interaction Submit Sample, page 353](#)

Overview

The following sample-code comparisons will help you modify your existing customized code to meet Genesys current recommendations and best-practice standards.

Common Code

The “Chat Sample” on [page 348](#) shows an example of how to use the `LoadBalancer.ServiceInfo(application type, tenant name)` method to populate a `ServiceInfo` object. This method is used in most of the samples. The only difference is that the application-type parameter would be specific to your needs. For instance, in the e-mail sample, the application type is `CfgAppType.CFGEmailServer`.

Another important difference to note is the use of protocols. The Platform SDK provides many different types of protocols. Review the following code selections to familiarize yourself with their implementation (see [page 349](#) and [page 352](#)).

The `KeyValueCollection` object is used in many of the samples to hold user data. In the past, this type of information was held in `_kvList` objects (or an object of similar type). An example of how to use the `KeyValueCollection` is found on [page 355](#).

Submissions to the server also have changed. Now, you first must create a `SubmitRequest` object. For examples of this, see [page 353](#) and [page 355](#).

After a submission is made to a server, you must check the response. Use the `EventStatus` or `EventAck` objects to determine the status of server connection request or the results of data-submission request. The following code snippets illustrate the use of `EventStatus` and `EventAck` objects (see [page 350](#) and [page 353](#)).

These highlights represent just a few of the code changes that exist in the new version of the samples. Please read the remainder of this chapter for more information that is specific to the changes. For more detailed information about the code that is contained within the samples, see Chapter 9, “eServices Samples for Java,” on [page 101](#) as well as the samples themselves.

Chat Sample

This section compares the older versions of the Chat Sample with the current 8.0 version. Please review the section “Common Code” on [page 347](#) for information about the commonalities between all of the Java samples.

Code Comparisons

The methods that are used to create an instance of load balancer and select a chat server have changed. In the 8.0 version of the samples, you use the `LoadBalancer.GetServiceInfo` method to obtain the `ServiceInfo` object information. Now, you can use the `ServiceInfo` methods to get your host, port, and alias information:

```
...else
{
    ServiceInfo si = LoadBalancer.GetServiceInfo(CfgAppType.CFGChatServer, strTenant);
    svcHost = si.getHost();
    svcPort = si.getWebApiPort();
    if(svcHost == null || svcHost.equalsIgnoreCase("") || svcPort == -1)
    {
        itf_response = "NOSERVICE";
        itf_message = "Chat service is unavailable at this time, please try again
            later.";
    }
    else
    {
        // we have a chat server at this point
        chat_alias = si.getAlias();
    }
}
```

In older versions of the code, you first declared an instance of the `SvcDispatcher` class and then called the `inqSrvcByType()` method. Then, you used the `SvcDispatcher`'s methods to get your host, port, and alias information:

```
try
{
    if (svcDispatcher == null)
        svcDispatcher = new SvcDispatcher();
}...

...else
{
    if( svcDispatcher == null || svcDispatcher.getErrorCode() != 0 ||
        !svcDispatcher.inqSrvcByType(CfgAppType.CFGChatServer, strTenant) )
    {
        itf_response = "NOSERVICE";
        itf_message = "Chat service is unavailable at this time, please try again
            later.";
    }
    else
    {
        // we have a chat server at this point
        chat_alias = svcDispatcher.getSvcAlias();
    }
}
```

The 8.0 version of the code makes use of one of the many protocols that are available in the Platform SDK. The following code illustrates usage of the `FlexChatProtocol`:

```
try
{
    FlexChatProtocol chat = new FlexChatProtocol (new Endpoint(new URI("tcp://" + svcHost
        + ":" + String.valueOf(svcPort))));
    chat.open();
}
```

In older versions, your code would have resembled something like the following:

```
_chat_direct chat = new _chat_direct(svcDispatcher.getSvcHost(),
    svcDispatcher.getSvcPort());
```

The new version also uses a `KeyValueCollection` object to store user data:

```
KeyValueCollection userdata = new KeyValueCollection();
userdata.addString(fldnFirstName, first_name);
userdata.addString(fldnLastName, last_name);...
```

In the past, this information would have been stored in a `_kvlist` object:

```
_kvlist userdata = new _kvlist();
userdata.addElement(new _kvitem(fldnFirstName, first_name));
userdata.addElement(new _kvitem(fldnLastName, last_name));...
```

The login request code also has changed. Notice the use of the `Event Status` object. Its methods are used to obtain `secure_key` and `user_id`:

```
RequestLogin rl = RequestLogin.create(strNickName, new Integer(8), userdata);
rl.setTimezoneOffset(Integer.parseInt(timeZoneOffset));

msg = chat.request(rl, 30000);
if (msg != null)
{
    if (msg.messageId() == EventStatus.ID)
    {
        EventStatus status = (EventStatus)msg;
        if (status.getReferenceId().intValue() != 0 && status.getSecureKey() != null)
        {
            secure_key = status.getSecureKey();
            user_id = status.getUserId();
            itf_response = "CONNECTED";
            itf_message = "Welcome to Genesys chat!";
        }
    }
}
```

The following shows the older version of the code:

```
int rc = chat.login(strNickName, userdata, timeZoneOffset);
if( _chat_direct.__rc_ok == rc )
{
    if( chat.user_id() != null && chat.secure_key() != null )
    {
        secure_key = chat.secure_key();
        user_id = chat.user_id();
        itf_response = "CONNECTED";
        itf_message = "Welcome to Genesys chat!";
    }
}
```

Also, note the differences in the following new sample with regard to the request to join code:

```
RequestJoin rj = RequestJoin.create(user_id, secure_key, "", strTenant + ":" +
    strQueueKey, subject);
msg = chat.request(rj, 30000);
EventStatus es = (EventStatus) msg;
```

The following shows the older code:

```
rc = chat.join(user_id, secure_key, strTenant + ":" + strQueueKey, subject);
if( _chat_direct.__rc_ok == rc && chat.user_id() != null && chat.secure_key() !=
```

```

    null && chat.transcript() != null )
{...

```

Another notable difference is the way in which you connect to the server the second time, by using the alias. The following shows how the alias is used today:

```
ServiceInfo si = LoadBalancer.GetServiceInfo(chat_alias);
```

The following shows the way in which you utilized the alias variable in the past:

```

if( svcDispatcher.inqSrvByAlias(chat_alias) )
{
    _chat_direct chat = new _chat_direct(svcDispatcher.getSrvHost(),
        svcDispatcher.getSrvPort());

```

There are code changes in the way in which you log out of the server. Now, you create a RequestLogout object:

```

RequestLogout rlo = RequestLogout.create(user_id, secure_key, new Integer(0));
msg = chat.request(rlo, 30000);

```

The following shows how the code used to appear:

```
chat.logout(user_id, secure_key);
```

The refresh code also has changed. Now, you create a RequestRefresh object:

```

RequestRefresh rr = RequestRefresh.create(user_id, secure_key, new
    Integer(Integer.parseInt(script_pos)+1) , MessageText.create("text", TreatAs.NORMAL,
    msg2send.equalsIgnoreCase("") ? null : msg2send));
msg = chat.request(rr, 30000)

```

The following shows how the code would have been written in older versions of the sample:

```

int rc = _chat_direct.__rc_ok;
if( msg2send.equals("") )
    rc = chat.refresh(script_pos, secure_key, user_id);
else
    rc = chat.refresh(script_pos, secure_key, user_id, msg2send);

```

E-Mail Sample

This section compares the older versions of the E-Mail Sample with the current 8.0 version. Please review the section “Common Code” on [page 347](#) for information about the commonalities between all of the Java samples.

Code Comparisons

As with the Chat Sample the E-Mail Sample utilizes a Platform SDK protocol. The following code illustrates usage of the `EmailProtocol`:

```
if (action != null && !action.equals(""))
{
    EmailProtocol email = new EmailProtocol(new Endpoint(new URI("tcp://" + svcHost + ":"
        + String.valueOf(svcPort))));
    boolean bConnected = false;
    try
    {
        email.open();
        bConnected = true;
    }
    catch (Exception e)
    {
```

The following shows how the code used to appear in older versions of the sample

```
if( action != null && !action.equals("") )
{
    _irs_direct irs = null;
    try
    {
        irs = new _irs_direct(svcHost, svcPort);
    }
    catch(_communication_exception cex)
    {
```

The new version uses an `EmailProperties` object to store the e-mail data:

```
if (bConnected && action.equals("Submit"))
{
    EmailProperties mProperties = new EmailProperties();
    mProperties.setFirstName(first_name);
    mProperties.setLastName(last_name);
    mProperties.setFromAddress(email_address);
    mProperties.setEmailBody(body);
    mProperties.setSubject(subject);
    if (reply_from != null && !reply_from.equalsIgnoreCase(""))
```

```
mProperties.setMailbox(reply_from);
```

In the past, this information would have been stored in a `_kvlist` object:

```
if( action.equals("Submit") )
{
    _kvlist userdata = new _kvlist();
    userdata.addElement(new _kvitem(fldnFirstName, first_name));
    userdata.addElement(new _kvitem(fldnLastName, last_name));
    userdata.addElement(new _kvitem(fldnFromAddress, email_address));
    userdata.addElement(new _kvitem(fldnSubject, subject));
    userdata.addElement(new _kvitem(fldnEmailBody, body));
    if (reply_from != null && reply_from.equals("") == false)
        userdata.addElement(new _kvitem(fldnReplyFrom, reply_from));
}
```

The submission is performed by using a `RequestSubmit` object, and the response is handled in an `EventAck` object in the new version:

```
RequestSubmit reqSubmit = RequestSubmit.create("email", mProperties);
Message msg = email.request(reqSubmit, 30000);

if (msg instanceof EventAck)
{
    EventAck eventAck = (EventAck) msg;
    id = eventAck.getRequestId();
}
```

In the older version the submission and response handling would have appeared like as follows:

```
if( irs.submit(_media.__media_email, userdata) == irs.__rc_ok )
{
    id = new String(irs.reqid());
}
```

Interaction Submit Sample

This section compares the older versions of the Interaction Submit Sample with the current 8.0 version. Please review the section “Common Code” on [page 347](#) for information about the commonalities between all of the Java samples.

Code Comparisons

The Interaction Submit Sample utilizes and number of new `LoadBalancer` methods to obtain the tenant ID and then to create a `KeyValueCollection` of application end points.

The following new method is called to obtain the tenant ID by using LoadBalancer:

```
String strTenantID = LoadBalancer.getTenantId(strTenant);
```

The following shows the older svcDispatcher method:

```
long longTenantID = svcDispatcher.getTenantId(strTenant);
```

Next, LoadBalancer methods are used to populate a KeyValueCollection with endpoint data:

```
kvcAppEndPoints = LoadBalancer.getOwnApp().getOptions().getList("endpoints:" +
    strTenantID);
try
{
    iTenantId = Integer.parseInt(strTenantID); ...
```

In older versions of the code, this same information would have been gathered by using svcDispatcher methods and stored in Hashtables:

```
if (longTenantID != null)
    iTenantID = longTenantID.intValue();
app = svcDispatcher.getOwnWebApplication();
Hashtable htAllOptions = app.getAllOptions();
if (htAllOptions != null)
    htAppEndpoints = (Hashtable) htAllOptions.get ("endpoints:" +
        longTenantID.toString()); ...
```

The Platform SDK also provides a protocol for Interaction Server. The following code demonstrates how to use it:

```
if(svcHost != null && !svcHost.equalsIgnoreCase("") && svcPort != -1 && strAction !=
    null && !strAction.equalsIgnoreCase(""))
{
    InteractionServerProtocol itxProtocol = new InteractionServerProtocol
        (new Endpoint(new URI("tcp://" + svcHost + ":" + String.valueOf(svcPort))));
    itxProtocol.setClientType(InteractionClient.MediaServer);
    itxProtocol.setClientName("MCR_WebAPIServer80");
```

The preceding code was represented in older versions as follows:

```
if (strAction != null && !strAction.equals(""))
{
    try
    {
        if(!strHost.equals(""))
        {
```

```

_interaction_direct itx = new _interaction_direct (strHost,
    Integer.parseInt(strPort));
rc = itx.register ("JSPSample");

```

Next, you create a `RequestSubmit` object and fill a `KeyValueCollection` with user data that is to be submitted later:

```

if (strAction != null && strAction.equals("Send") && bConnected)
{
    try
    {
        RequestSubmit reqSubmit = RequestSubmit.create();
        KeyValueCollection kvcUserData = new KeyValueCollection();
        kvcUserData.addString(attachDataName1, attachDataValue1);
        kvcUserData.addString(attachDataName2, attachDataValue2);
        kvcUserData.addString(attachDataName3, attachDataValue3);
        kvcUserData.addString("FirstName", strFirstName);
        kvcUserData.addString("LastName", strLastName);...
    }
}

```

Older versions of the code gathered this information in a `TKVList` object:

```

if (strAction != null && strAction.equals("Send") && bError == false)
{
    try
    {
        com.genesyslab.commons.collections.TKVList user_data =
            new com.genesyslab.commons.collections.TKVList();
        user_data.addString (attachDataName1, attachDataValue1);
        user_data.addString (attachDataName2, attachDataValue2);
        user_data.addString (attachDataName3, attachDataValue3);
        user_data.addString ("FirstName", strFirstName);
        user_data.addString ("LastName", strLastName);...
    }
}

```

Here, you set the properties of your recently created `RequestSubmit` object:

```

reqSubmit.setUserData(kvcUserData);
reqSubmit.setMediaType(strMediaType);
reqSubmit.setInteractionId(strInteractionID);
reqSubmit.setQueue(strScriptName);
reqSubmit.setTenantId(iTenantId);
reqSubmit.setInteractionType("Inbound");
reqSubmit.setInteractionSubtype("InboundNew");
reqSubmit.setIsOnline(Boolean.FALSE);

```

Then, you make the submission:

```

Message msg = itxProtocol.request(reqSubmit, 30000);

```

In older versions of the sample, the code would have appeared as follows:

```
if (longTenantID != null)
{
    iTenantID = longTenantID.intValue();
    rc = itx.submit(strMediaType, strInteractionID, strScriptName, iTenantID, user_data);
```

In the new sample, the methods of the EventAck that you receive in response to the submission are used to obtain the interaction ID:

```
if (msg != null && msg instanceof EventAck)
{
    EventAck eventAck = (EventAck) msg;
    strInteractionID = eventAck.getExtension().getString("InteractionId");
    strError = "Operation successfully submitted to the " + svcHost + ":" + svcPort;...
```

In the following, you will see that the older code versions obtained the interaction ID by using the `_interaction_direct.get_interaction_id()` method:

```
if (rc != _interaction_direct.__rc_ok)
{
    strError = itx.lasterror() + "\r\n" + svcDispatcher.getErrorMsg();
    bError = true;
}
//If InteractionID was generated by server.
strInteractionID = itx.get_interaction_id();...
```

Updates also are coded differently. In the new version of the sample, you first create a RequestChangeProperties object:

```
else if (strAction != null && strAction.equals("Update") && bError == false)
{
    RequestChangeProperties requestChangeProperties = RequestChangeProperties.create();
```

Then, you create a KeyValueCollection to hold the data that you are removing:

```
KeyValueCollection kvcDeletedUserData = new KeyValueCollection();
kvcDeletedUserData.addString(attachDataName3, attachDataValue3);
```

In older versions, this data would have been stored in a TKVList object:

```
com.genesyslab.commons.collections.TKVList deleted_user_data =
    new com.genesyslab.commons.collections.TKVList();
deleted_user_data.addString(attachDataName3, attachDataValue3);
```

In the new version, a second `KeyValueCollection` holds the new data:

```
KeyValueCollection kvcUpdatedUserData = new KeyValueCollection();
kvcUpdatedUserData.addString(attachDataName1, attachDataValue1);
kvcUpdatedUserData.addString(attachDataName2, attachDataValue2);
kvcUpdatedUserData.addString("FirstName", strFirstName);
kvcUpdatedUserData.addString("LastName", strLastName);...
```

In older versions, however, the information for updating would be contained in a `TKVList` object:

```
com.genesyslab.commons.collections.TKVList user_data =
    new com.genesyslab.commons.collections.TKVList();
user_data.addString (attachDataName1, attachDataValue1);
user_data.addString (attachDataName2, attachDataValue2);
user_data.addString ("FirstName", strFirstName);
user_data.addString ("LastName", strLastName);
```

Then, the new version of the code calls `RequestChangeProperties` methods to remove old data and add the updates:

```
requestChangeProperties.setInteractionId(strInteractionID);
requestChangeProperties.setAddedProperties(kvcUpdatedUserData);
requestChangeProperties.setDeletedProperties(kvcDeletedUserData);
```

Finally, the new code passes the `RequestChangeProperties` object into the request:

```
Message msg = itxProtocol.request(requestChangeProperties, 30000);
```

The following shows how this same code would have appeared in older versions of the sample:

```
rc = itx.change_properties(strInteractionID, user_data, deleted_user_data);
```

The cancel code also is new. The new code requires that a `ReasonInfo` object be passed into the `RequestStopProcessing` object:

```
else if (strAction != null && strAction.equals("Cancel") && bError == false)
{
    ReasonInfo reason = ReasonInfo.create();
    reason.setReason(123456789);
    reason.setReasonDescription("Put your reason here");
```

In the following, you will see where the `ReasonInfo` object that was just created is passed in:

```
RequestStopProcessing requestStopProcessing =  
    RequestStopProcessing.create(strInteractionID, reason);
```

Then, you simply pass the RequestStopProcessing into the request method:

```
Message msg = itxProtocol.request(requestStopProcessing, 30000);
```

In older versions of the code, the request would have appeared as follows:

```
else if (strAction != null && strAction.equals("Cancel") && bError == false)  
{  
    try  
    {  
        rc = itx.stop_processing(strInteractionID, 12345, "Put your reason here"); ...
```



Appendix

B

Statistical Information Sample Configuration

This chapter explains the `webapistats.cfg` file. Statistical Information Sample uses this file to provide statistical information.

This appendix contains the following sections:

- [The webapistats.cfg File, page 359](#)

The webapistats.cfg File

The following is a detailed description of the contents of the `webapistats.cfg` file, which the Statistical Information Sample uses to provide statistical information.

[WebApiSrvQueue_chat_Current_In_Queue]

AggregationType=Current

Category=JavaCategory

Description=Number of chat interactions in the queue at the moment of measurement

JavaSubCategory=eServiceInteractionStat.jar:OMQ Current in Queue

MediaType=chat

Objects=StagingArea

[WebApiSrvQueue_chat_Current_In_Processing_In_Queue]

AggregationType=Current

Category=JavaCategory

Description=Number of chat interactions that have been submitted to a Queue and are currently being processed

JavaSubCategory=eServiceInteractionStat.jar:OMQ Current in Processing
MediaType=chat
Objects=StagingArea

[WebApiSrvQueue_chat_Current_Waiting_Processing_In_Queue]

AggregationType=Current
Category=JavaCategory
Description=Number of chat interactions that have been submitted to a Queue
and are currently awaiting processing
JavaSubCategory=eServiceInteractionStat.jar:OMQ Current Waiting
Processing
MediaType=chat
Objects=StagingArea

[WebApiSrvQueue_chat_Total_Distributed]

AggregationType=Total
Category=JavaCategory
Description=Total number of chat interactions that have been distributed to
Agents
JavaSubCategory=eServiceInteractionStat.jar:cs total distributed chat
Objects=StagingArea

[WebApiSrvQueue_chat_Total_Distribution_Time]

AggregationType=Total
Category=JavaCategory
Description=Total time that chat interactions have been awaiting distribution
JavaSubCategory=eServiceInteractionStat.jar:cs total distribution time chat
Objects=StagingArea

[WebApiSrvQueue_email_Queue_Length]

AggregationType=Current
Category=JavaCategory
Description=Total number of e-mail/webform interactions that are in queue at
the requested moment and have not been distributed to Agents
JavaSubCategory=eServiceInteractionStat.jar:cs queue length webform
Objects=StagingArea

[WebApiSrvQueue_email_Total_Distributed]

AggregationType=Total

Category=JavaCategory

Description=Total number of e-mail/webform interactions that have been distributed to Agents

JavaSubCategory=eServiceInteractionStat.jar:cs total distributed webform

Objects=StagingArea

[WebApiSrvQueue_email_Total_Waiting_Time]

AggregationType=Total

Category=JavaCategory

Description=Total number of seconds that e-mail/webform interactions have been waiting to be processed (or abandoned)

JavaSubCategory=eServiceInteractionStat.jar:OMQ Total Waiting Time

MediaType=email



Supplements

Related Documentation Resources

The following resources provide additional information that is relevant to this software. Consult these additional resources as necessary.

eServices Web API

- *eServices 8.0 Deployment Guide*, which introduces the architecture, required components, and procedures for deploying eServices.
- *eServices 8.0 Reference Manual*, which provides a reference listing of all configuration options and of field codes used in standard responses.
- *eServices 8.0 User's Guide*, especially its Load Balancing chapter.
- *eServices 8.0 Universal Contact Server Manager Help*, which is a guide to the user interface for Universal Contact Server Manager.
- *eServices 8.0 Knowledge Manager Help*, which is a guide to the Knowledge Manager user interface.
- *eServices 8.0 Interaction Workflow Designer Help*, which is a guide to the user interface for Interaction Workflow Designer.
- *eServices Social Media Solution Guide*, which provides information about deploying and using the Genesys Social Messaging Management product. It is available on the Genesys Documentation Wiki at http://developerzone.genesyslab.com/wiki/index.php?title=Category:EServices_Social_Media_Solution_Guide
- *Platform SDK 8.0 Java (or .NET) API Reference*.
These references list the classes, methods, fields, and constants of the Web API portion of the Web API Server component. All of these references are available on the Genesys DevZone portal, and also located on the Genesys Developer Documentation Library DVD.
- “eServices Log Events” in *Framework 8.0 Combined Log Events Help*, which is a comprehensive list and description of all events that may be recorded in logs.

- *KANA Response Live/Hipbone Design Guidelines: Synetry Cobrowse*, which offers design recommendations for co-browsing applications. Genesys redistributes this document from KANA Software, Inc., which licenses certain KANA Response Live (formerly Hipbone) cobrowsing components to Genesys. The document is available on the Genesys Technical Support website at <http://genesyslab.com/support>.
- *KANA Response Live/Hipbone Client API Reference Guide*, which provides integration details for co-browsing applications. Genesys redistributes this document from KANA Software, Inc., which licenses certain KANA Response Live (formerly Hipbone) cobrowsing components to Genesys. The document is available on the Genesys Technical Support website at <http://genesyslab.com/support>.

Genesys

- *Genesys Technical Publications Glossary*, which ships on the Genesys Documentation Library DVD and which provides a comprehensive list of the Genesys and computer-telephony integration (CTI) terminology and acronyms used in this document.
- *Genesys Migration Guide*, which ships on the Genesys Documentation Library DVD, and which provides documented migration strategies for Genesys product releases. Contact Genesys Technical Support for more information.
- Release Notes and Product Advisories for this product, which are available on the Genesys Technical Support website at <http://genesyslab.com/support>.

Information about supported hardware and third-party software is available on the Genesys Technical Support website in the following documents:

- *Genesys Supported Operating Environment Reference Manual*
- *Genesys Supported Media Interfaces Reference Manual*

Consult these additional resources as necessary:

- *Genesys 8 Events and Models Reference Manual*, which includes a set of basic interaction models, showing the components involved and the event messages sent among them. These models and events were formerly presented in the *Genesys 7 Events and Models Reference Manual*. The request messages that were also described in that book are now documented in the API References of the Platform SDK.
- The documentation on the other three members of the Genesys Customer Interaction Platform: Universal Routing, Reporting, and Management Framework.

For additional system-wide planning tools and information, see the release-specific listings of System Level Documents on the Genesys Technical Support website, accessible from the [system level documents by release](#) tab in the Knowledge Base Browse Documents Section.

Genesys product documentation is available on the:

- Genesys Technical Support website at <http://genesyslab.com/support>.
- Genesys Documentation Library DVD, which you can order by e-mail from Genesys Order Management at orderman@genesyslab.com.

Document Conventions

This document uses certain stylistic and typographical conventions—introduced here—that serve as shorthands for particular kinds of information.

Document Version Number

A version number appears at the bottom of the inside front cover of this document. Version numbers change as new information is added to this document. Here is a sample version number:

80fr_ref_06-2008_v8.0.001.00

You will need this number when you are talking with Genesys Technical Support about this product.

Screen Captures Used in This Document

Screen captures from the product graphical user interface (GUI), as used in this document, may sometimes contain minor spelling, capitalization, or grammatical errors. The text accompanying and explaining the screen captures corrects such errors *except* when such a correction would prevent you from installing, configuring, or successfully using the product. For example, if the name of an option contains a usage error, the name would be presented exactly as it appears in the product GUI; the error would not be corrected in any accompanying text.

Type Styles

[Table 13](#) describes and illustrates the type conventions that are used in this document.

Table 13: Type Styles

Type Style	Used For	Examples
Italic	- Document titles - Emphasis - Definitions of (or first references to) unfamiliar terms - Mathematical variables Also used to indicate placeholder text within code samples or commands, in the special case where angle brackets are a required part of the syntax (see the note about angle brackets on page 368).	Please consult the <i>Genesys Migration Guide</i> for more information. Do <i>not</i> use this value for this option. A <i>customary and usual</i> practice is one that is widely accepted and used within a particular industry or profession. The formula, $x + 1 = 7$ where x stands for . . .
Monospace font (Looks like teletype or typewriter text)	All programming identifiers and GUI elements. This convention includes: - The <i>names</i> of directories, files, folders, configuration objects, paths, scripts, dialog boxes, options, fields, text and list boxes, operational modes, all buttons (including radio buttons), check boxes, commands, tabs, CTI events, and error messages. - The values of options. - Logical arguments and command syntax. - Code samples. Also used for any text that users must manually enter during a configuration or installation procedure, or on a command line.	Select the Show variables on screen check box. In the Operand text box, enter your formula. Click OK to exit the Properties dialog box. T-Server distributes the error messages in EventError events. If you select true for the inbound-bsns-calls option, all established inbound calls on a local agent are considered business calls. Enter exit on the command line.
Square brackets ([])	A particular parameter or value that is optional within a logical argument, a command, or some programming syntax. That is, the presence of the parameter or value is not required to resolve the argument, command, or block of code. The user decides whether to include this optional information.	<pre>smcp_server -host [/flags]</pre>

Table 13: Type Styles (Continued)

Type Style	Used For	Examples
Angle brackets (< >)	A placeholder for a value that the user must specify. This might be a DN or a port number specific to your enterprise. Note: In some cases, angle brackets are required characters in code syntax (for example, in XML schemas). In these cases, italic text is used for placeholder values.	<code>smcp_server -host <confighost></code>



Index

Symbols

[] (square brackets)	367
< > (angle brackets)	368

A

abandon	69
acknowledgement	
Genesys 3rd party media	72
advanced chat	
sample	36, 132
alias request	68
alias, chat	68
angle brackets	368
API Access	
FAQ sample	38
application	27
Asian languages	33
audience, for document	16
authentication	344, 345

B

basic cobrowse	
sample	309–314
binary data	26
blank.html	54, 55, 56, 57, 58, 197, 204, 308, 315
brackets	
angle	368
square	367
browser, identifying	105, 229

C

callback	
sample	53
category	82, 83
character encoding	107
character set	34

character set, multi-byte	33
charset option	34
chat	16, 25
advanced	47, 51
alias	68
and cobrowse	51
and cobrowse sample	55
and cobrowse sample, files	55
API	65
architecture	65
concepts	65
error	66
exception	66
join request	67
logging in	66
login request	67
logout request	67
packet status	130, 249, 260
refresh request	67, 69
sample	36, 46, 51, 238–251, 251–262
sample, files	51
sample, JavaScript functions	132, 241
Server	65, 66, 67, 125
Server, disconnecting	68
service	65–70
session	66, 67, 127
session, event flow	67–68
session, life cycle	66
state diagram	66
transcript	67, 69
transcript, empty	69
UI implementation	239, 252
user interface implementation	122
user requests	127
chat and cobrowse	38
sample	47, 315
Chat Server. See chat	
chat widget	
sample	52, 134
chat with AJAX	
sample	52

chatandcobrowse.htm 204
 chatselftest.jsp 58
 CIM (Customer Interaction Management)
 Platform 14
 class hierarchy 26
 classes
 _chat_direct 70
 _chat_transcript 70
 _irs_direct 63
 classification 61, 63
 Classification Server 72, 83
 client application 16, 25
 client scenarios 343–346
 cobrowse
 sample 47, 54
 sample, files 54
 samples 54
 cobrowse integration support files 78
 cobrowse overview
 sample 308–309
 cobrowse with dynamic startup page 37
 sample 47, 56, 330–333
 sample, files 56
 cobrowse with initial start page
 sample, files 56
 cobrowse with initial startup page
 sample 47, 56, 326–329
 cobrowse with meet me 38
 sample 47, 57, 322–326
 sample, files 57
 cobrowse.htm 55
 code page option 34
 commenting on this document 18
 commlib.js 104, 105, 229
 communication channels 71
 Configuration Server 28
 configuration settings 27
 connect 69
 constants 105, 229
 constants.cs 228, 229
 constants.jsp 104
 contact centers 71
 conventions
 in document 366
 type styles 367
 custom application 72
 customer-relationship management (CRM) 71

D

diagnostic tests 28
 disconnecting
 Chat Server 67, 68
 document
 audience 16
 change history 19

conventions 366
 errors, commenting on 18
 version number 366

E

e-mail 16, 25, 53
 error 111, 118, 273
 event flow 62–64
 sample 35, 37, 46, 53, 108–113, 269–274
 sample, files 53
 sample, JavaScript functions 113, 120, 270
 Server Java 62, 63, 109, 113, 120, 269
 service, components 61
 session, life cycle 62
 state diagram 62
 e-mail aspx files 270
 E-mail Server Java. See e-mail
 e-mail with attachment
 sample 46, 53
 e-mail with attachment sample 113
 files 53
 e-mail, web safe 180
 email.jsp 53
 emailselftest.jsp 58
 empty chat transcript 69
 empty state 66
 envelope 26, 27
 envelope factory 27
 error
 chat 66
 e-mail 111, 118, 273
 Genesys 3rd party media sample 141, 154, 266
 Interaction Server 141, 154, 266
 eServices Interactive Management CD 102, 226
 eServices test tools 28
 event flow
 chat session 67–68
 e-mail 62–64
 Genesys 3rd party media 72
 event_body 70
 event_type 69
 exampleofdynamicstartpage.jsp 219, 330
 exception
 chat 66
 Genesys 3rd party media sample 141, 154, 266
 Interaction Server 141, 154, 266

F

FAQ
 sample 47
 FAQ sample
 FAQ 16, 25
 FAQ.jsp 57

- files 57
- Java 57
- fax 71
- Flex Chat. See chat
- flex-disconnect-timeout. 67
- fon.gif. 104
- font styles
 - italic 367
 - monospace 367
- form parameters, retrieving 105, 229
- frame handles, retrieving 105, 229

G

- Genesys 3rd party media 25, 26, 71–72
 - acknowledgement 72
 - event flow 72
 - sample. 47, 53, 151–159, 274–285
 - sample, files 53
 - sample, JavaScript functions 278
- Genesys 3rd party media sample, JavaScript functions 159
- Genesys Configuration Server. See Configuration Server
- Genesys Interaction Server. See Interaction Server
- genesyslogo-trans.gif 104
- getAlias. 30
- getEventType
 - method 130
- getHost. 30
- getServiceInfo 30
- getServiceInfo(strAliasName) 30
- getWebApiPort 30
- graphical user interface
 - samples 107, 230

H

- hbapi.html 54, 55, 56, 57, 197, 204, 308, 315
- hbmessage_to_var.html 54, 55, 56, 57, 197, 204, 308, 315
- hbmessage_to_var.js. 54, 55, 56, 57, 197, 204, 308, 315
- hbmessaging.js. 55, 56, 57, 197, 204, 308, 315
- hbmessagingform.html 55, 56, 57, 197, 204, 308, 315
- hidden variables 131
- host. 63
- host/port
 - resolve alias 68
- HTML form parameters, retrieving 105, 229
- HTML frame handles, retrieving 105, 229
- htmlchatcommand.jsp 51, 122, 133, 204
- htmlchatframeset.jsp 51, 122, 133, 204

- htmlchatpanel.jsp 51, 122, 133, 204
- http request 68
- HttpServletRequest 34
- HttpServletResponse 34

I

- i18nsupport 33
- ID
 - session 66
- index.html 58, 104
- initialstartpageexample.html 215, 326
- installation testing 48
- Installing samples. See samples
- intelligent routing. 61
- intended audience 16
- interaction 26, 71–72
- Interaction Server 63, 71–72
 - error. 141, 154, 266
 - exception 141, 154, 266
 - host 274
 - port 274
- interaction submit sample. See Genesys 3rd party media sample
- international language support. 33
- italics. 367
- itxsubmit.aspx files. 276
- itxsubmit.jsp 53, 153

J

- Javadoc
 - Platform SDK API Reference 102
- JavaScript 105, 229
 - common functions 105, 229
 - functions, chat sample 132, 241
 - functions, e-mail sample 113, 120, 270
 - functions, Genesys 3rd party media sample 159, 278
- join method 66
- JSP 63

K

- knowledge management. 61, 63

L

- last_position 69
- lbtestpage.jsp 58
- life cycle 62, 66
- load balancing 28–33, 229
 - servlet. 110, 116, 141, 154, 198, 205, 266, 272, 310, 316

localization 33
 logged in state 66
 login method 70

M

mask_html method 107
 masking text 107
 MaskSymbols
 method 129, 249, 261
 MBCS (multi-byte character set) 33
 media channels 14
 media type 71, 274
 message 69
 monospace font 367
 multi-byte character set 33

O

option settings 27

P

package structure 26
 packages. *See also individual package names* 26
 packet 26
 party_id 70
 Platform SDK API Reference (Javadoc) . . . 102
 port 63

Q

qstring.js 55, 56, 57, 197, 204, 308, 315

R

refresh
 request, chat 69
 reporting 71
 resolve alias, host/port 68
 responseLive.js 56
 responseLive.StartApp.html 56
 responseLiveLauncher.html 56
 responseLiveScriptletLauncher.html 56
 routing 61, 63
 Routing Server, Universal 109, 269

S

samples 34, 45–59, 225
 .NET 225–308
 directory structures 49
 files 50

installing 48
 Java 101–223
 shared files 104, 228
 samples. *See individual samples by name*
 scenarios, client 343–346
 secure e-mail 180
 security.jsp 104, 107
 server event 130, 249, 260
 ServiceInfo 30, 30–33
 servlet container 25
 servlet engine 25
 session_id 69
 session_id parameter 66
 session, chat 66
 setContentType method 34
 square brackets 367
 start_at 69
 Stat Server 286
 state 62
 diagram, chat 66
 diagram, e-mail 62
 statinfo.aspx files 286
 statistics 61
 Statistics Server 160
 strategies 61, 63
 string manipulation and
 encoding function 105, 229
 survey
 sample 52, 138–144, 262–269
 sample, files 52
 survey.jsp 52

T

test tools 48, 335
 files 58
 time_offset 70
 time, retrieving 105, 229
 timeout 67
 Tomcat 49
 Training Server 83
 transcript method 70
 transcript, chat 67, 69
 troubleshooting guide 341
 type styles
 conventions 367
 italic 367
 monospace 367
 typographical styles 367

U

UI implementation, chat 122
 Unicode 34
 Universal Contact Server

sample. 166–181, 294–308
 Universal Routing Server. 109, 269
 user behavior, tracking 345–346
 user interface implementation, chat . . . 239, 252
 user requests, chat. 127
 user_nick. 70
 user_type. 70

V

vector. 70
 version numbering, document 366

W

Web API
 architecture 25
 architecture, e-mail 26
 class hierarchy 26
 JSP 63
 package structure 26
 Server 71, 72
 servlet 63
 usage 34
 web browser 72
 web callback
 sample. 53, 144–151, 230–237
 web callback service 85–99
 web collaboration. 16, 25
 web form 71
 web safe e-mail 180
 web server 72
 web-based chat with AJAX. *See* chat
 web-based chat. *See* chat
 web-based e-mail. *See* e-mail
 web-based Genesys 3rd party media. *See*
 Genesys 3rd party media
 web-based history. *See* history
 web-based statistics. *See* statistics
 webcallback.aspx 231
 webcallback.jsp 53, 145
 WebLogic. 25
 WebSphere. 25
 workflow strategies. 61

X

XML 26

