Genesys Multimedia 7.6

Web API Client

Developer’s Guide

About Genesys

Genesys Telecommunications Laboratories, Inc., a subsidiary of Alcatel-Lucent, is 100% focused on software for call centers. Genesys recognizes that better interactions drive better business and build company reputations. Customer service solutions from Genesys deliver on this promise for Global 2000 enterprises, government organizations, and telecommunications service providers across 80 countries, directing more than 100 million customer interactions every day. Sophisticated routing and reporting across voice, e-mail, and Web channels ensure that customers are quickly connected to the best available resource—the first time. Genesys offers solutions for customer service, help desks, order desks, collections, outbound telesales and service, and workforce management. Visit 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

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 following regional numbers:

Region	Telephone	E-Mail
North and Latin America	+888-369-5555 or +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
Japan	+81-3-6361-8950	support@genesyslab.co.jp

Prior to contacting technical support, please refer to the *Genesys Technical Support Guide* for complete contact information and procedures.

Ordering and Licensing Information

Complete information on ordering and licensing Genesys products can be found in the ***Genesys 7 Licensing Guide*.**

Released by

Genesys Telecommunications Laboratories, Inc. www.genesyslab.com

Document Version: 76mm_dev_web-api_01-2009_v7.6.101.00

Table of Contents

Preface

Welcome to the *Multimedia 7.6 Web API Client Developer's Guide.* This book 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 Open Media services to submit web-based interactions of virtually any type to Genesys Interaction Server.

This guide is valid only for the 7.6 release(s) of this product.

Note: For versions of this guide created for other releases of this product, please 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 chapter contains these sections:

Multimedia and the CIM Platform

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

Genesys Multimedia (formerly Multi-Channel Routing) 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).

Multimedia 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.

Multimedia

Multimedia, 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 Multimedia because it deals with both traditional telephonic interactions and the nontraditional interactions that are handled in Multimedia.

- From the media channels, at least one of the following:
  - Genesys E-mail—e-mail
  - Genesys Web Media—chat
  - Genesys Open Media—ability to add customized support for other media (fax, for example)
- Optionally, Web Collaboration—the ability for agents and customers to co-browse (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 comprise Genesys Multimedia.

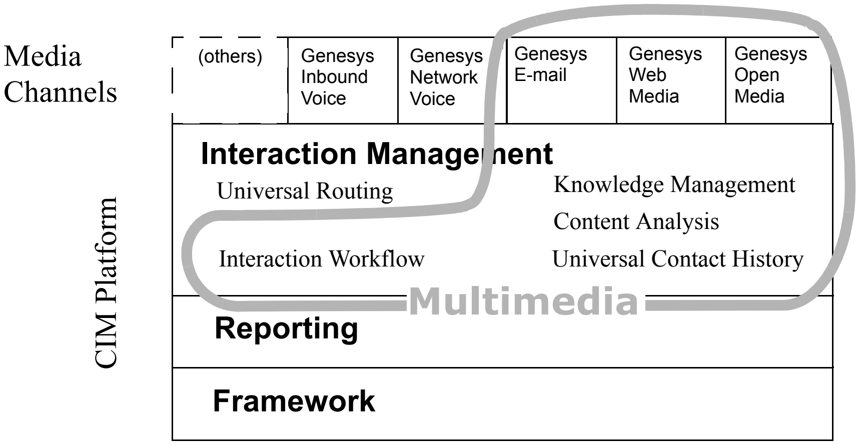


Figure 1: Multimedia in Relation to the CIM Platform and Media Channels

Note: Although Universal Routing is not considered part of Multimedia, 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 Resources" on page 17.

Licensing

Licensing requirements are:

- For each agent: one Multimedia 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 7 Licensing Guide.*

Reporting

Reporting templates are available for Multimedia. For details, see the *Reporting Technical Reference Guide for the Genesys 7.x Release.*

Intended Audience

This guide, primarily intended for developers, assumes that you have a basic understanding of:

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

You should also be familiar with:

- Genesys Framework architecture and functions.
- Genesys Multimedia 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:

- Server-side integration between Genesys software and third-party software.
- Creation of specialized web client applications that are specific to customer needs.

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

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 Genesys's current 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.

Chapter Summaries

In addition to this preface, this guide contains the following chapters:

- Chapter 1, "About Web API Clients," on page 21, puts the Multimedia Web API in a development context by outlining its architecture and discussing the main elements of its class hierarchy.
- Chapter 2, "About the Samples," on page 51, briefly describes the sample applications provided on the Multimedia Interaction Management CD and tells you how to install them.
- Chapter 3, "Understanding and Using the E-Mail Service," on page 67, provides an overview of the Multimedia e-mail service, including the life cycle and event flow of an e-mail.
- Chapter 4, "Understanding and Using the Flex Chat Service," on page 71, introduces the Multimedia Flex Chat service, including its life cycle and event flow.
- Chapter 5, "Understanding and Using the Callback Service," on page 77, introduces the Multimedia Voice Callback service, including its life cycle and event flow.
- Chapter 6, "Understanding and Using the Open Media Service," on page 81, introduces the Multimedia Open Media service, including its life cycle and event flow.
- Chapter 7, "Understanding and Using the Web Collaboration Service," on page 85, introduces the Web Collaboration service.
- Chapter 8, "Understanding and Using the FAQ Service," on page 91, introduces the FAQ object.
- Chapter 9, "Multimedia Simple Samples for Java," on page 95, reviews the simple callback, chat, co-browse, e-mail, and Open Media samples provided for Java developers on the Multimedia Interaction Management CD.
- Chapter 10, "Multimedia Simple Samples for .NET," on page 169, reviews the simple callback, chat, e-mail, and Open Media samples provided for .NET developers on the Multimedia Interaction Management CD.

- Chapter 11, “Multimedia Compound Sample,” on page 241, reviews the Compound Sample provided for Java developers on the Multimedia Interaction Management CD.
- Chapter 12, “Multimedia Test Tools,” on page 263, provides an overview of the test tools provided on the Multimedia Interaction Management CD and basic troubleshooting tips.
- Chapter 13, “Sample Client Scenarios,” on page 271, presents common scenarios for web client application development and shows how to satisfy them by using code snippets from the web samples.

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:

mm_dev_web-api_11-2007_v7.6.000.00

You will need this number when you are talking with Genesys Technical Support about this product.

Type Styles

Italic

In this document, italic is used for emphasis, for documents’ titles, for definitions of (or first references to) unfamiliar terms, and for mathematical variables.

Examples:

- Please consult the *Genesys 7 Migration Guide* for more information.
- *A customary and usual practice* is one that is widely accepted and used within a particular industry or profession.
- Do *not* use this value for this option.
- The formula, $x + 1 = 7$ where x stands for . . .

Monospace Font

A monospace font, which looks like `teletype or typewriter text`, is used for all programming identifiers and GUI elements.

This convention includes the *names* 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; and code samples.

Examples:

- Select the `Show variables on screen` check box.
- Click the `Summation` button.
- In the `Properties` dialog box, enter the value for the host server in your environment.
- In the `Operand` text box, enter your formula.
- Click `OK` to exit the `Properties` dialog box.
- The following table presents the complete set of error messages T-Server® distributes 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.

Monospace is also used for any text that users must manually enter during a configuration or installation procedure, or on a command line:

Example:

- Enter `exit` on the command line.

Screen Captures Used in This Document

Screen captures from the product GUI (graphical user interface), as used in this document, may sometimes contain a minor spelling, capitalization, or grammatical error. 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.

Square Brackets

Square brackets indicate that a particular parameter or value is optional within a logical argument, a command, or some programming syntax. That is, the parameter's or value's presence is not required to resolve the argument, command, or block of code. The user decides whether to include this optional information. Here is a sample:

```
smcp_server -host [/flags]
```

Angle Brackets

Angle brackets indicate a placeholder for a value that the user must specify. This might be a DN or port number specific to your enterprise. Here is a sample:

```
smcp_server -host <confighost>
```

Related Resources

Consult these additional resources as necessary:

- *Multimedia 7.6 Deployment Guide,* which introduces the architecture, required components, and procedures for deploying Multimedia.
- *Multimedia 7.6 Reference Manual,* which provides a reference listing of all configuration options and of field codes used in standard responses.
- *Multimedia 7.6 User's Guide,* especially its Load Balancing chapter.
- *Genesys 7 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 *Open Media Interaction Models Reference Manual*. The request messages that were also described in that book are now documented in the API References of the Platform SDK.
- *Multimedia 7.6 Universal Contact Server Manager Help,* which is a guide to the user interface for Universal Contact Server Manager.
- *Multimedia 7.6 Knowledge Manager Help,* which is a guide to the Knowledge Manager user interface.
- *Multimedia 7.6 Interaction Workflow Designer Help,* which is a guide to the user interface for Interaction Workflow Designer.
- *Multimedia 7.6 Web API Reference* (Javadoc), for the Java implementation; or its equivalents (in compiled HTML format) for the .NET implementation:
  - *Web Media Platform SDK 7.6 .NET API Reference.*
  - Other pertinent Platform SDK API References, notably the *Management Platform SDK .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.
- "Multimedia Log Events" in *Framework 7.6 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) co-browsing components to Genesys.
- *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) co-browsing components to Genesys.
- The *Genesys Technical Publications Glossary,* which ships on the Genesys Documentation Library DVD and which provides a comprehensive list of the Genesys and CTI terminology and acronyms used in this document.
- The *Genesys 7 Migration Guide,* also on the Genesys Documentation Library DVD, which provides a documented migration strategy from Genesys product releases 5.1 and later to all Genesys 7.x releases. Contact Genesys Technical Support for additional information.
- The Release Notes and Product Advisories for this product, which are available on the Genesys Technical Support website at `http://genesyslab.com/support`.
- The documentation on the other three members of the Genesys Customer Interaction Platform: Universal Routing, Reporting, and Management Framework.

Information on supported hardware and third-party software is available on the Genesys Technical Support website in the following documents:

- *Genesys Supported Operating Systems and Databases*
- *Genesys Supported Media Interfaces*

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`.

Making Comments on This Document

If you especially like or dislike anything about this document, please 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 information in this document only and to the way in which the information is presented. Speak to 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.

Document Change History

This section lists topics that are new in the current release of this document, or that have changed significantly from the preceding release.

New in Version 7.6.1

The following topics have been added or significantly changed since the initial 7.6 release:

- The `LoadBalancer.GetServiceInfo(ConfServerClientType, String)` method has been deprecated. It has been replaced by the `LoadBalancer.GetServiceInfo(CfgAppType, String)` method. The code snippets in Chapter 10, "Multimedia Simple Samples for .NET," on page 169 have been updated to include this change.
- Chapter 11, "E-Mail," on page 256 includes new functionality that enables us to link the reply to an e-mail that has its parent message stored in the Universal Contact Server Database.

Chapter

1 About Web API Clients

The Multimedia Web API enables you to write client applications that support chat, e-mail, web collaboration, voice callback, and FAQ interactions. It includes predefined, media-specific packages for these purposes. Using Open Media, you can also write applications that handle custom media types.

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

Note: To use the 7.6.1 version of Web API Server, you must also install the 7.6.1 version of the Third Party components.

Important Note About .NET Architecture

The package structure described in most of this chapter does not apply to the .NET implementation—it is specific to the Java version.

For details about the .NET implementation's architecture, refer to these sources:

- *Web Media Platform SDK .NET API Reference.*
- *Management Platform SDK .NET API Reference.*
- Other API References for the Platform SDK, as pertinent to the applications you are building.

For instructions about accessing these documents, see “Related Resources” on page 17.

Instead of the Java package structure shown in Figure 2, the .NET implementation contains a single “flat” directory, providing the following libraries:

- `Genesyslab.Core.dll`
- `Genesyslab.Platform.Commons.dll`
- `Genesyslab.Platform.Commons.Collections.dll`
- `Genesyslab.Platform.Commons.Collections.Binding.dll`
- `Genesyslab.Platform.Commons.Connection.dll`
- `Genesyslab.Platform.Commons.Protocols.dll`
- `Genesyslab.Platform.Configuration.Protocols.dll`
- `Genesyslab.Platform.Contacts.Protocols.dll`
- `Genesyslab.Platform.Management.Protocols.dll`
- `Genesyslab.Platform.OpenMedia.Protocols.dll`
- `Genesyslab.Platform.Outbound.Protocols.dll`
- `Genesyslab.Platform.Reporting.Protocols.dll`
- `Genesyslab.Platform.Voice.Protocols.dll`
- `Genesyslab.Platform.WebMedia.Protocols.dll`
- `LogLib4NET.dll`

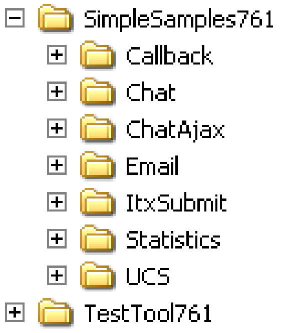


Figure 2: Web API Package Structure for .NET

Java Architecture

The Web API enables your customized server pages (JSPs or ASPXs) to perform common functions such as sending an e-mail or conducting a chat session. This is done by communicating with a servlet container—such as WebSphere, WebLogic, Tomcat, or JRun—that is running on a web server. The servlet container processes the JSP or ASPX and forwards the content to the appropriate server. Figure 3 shows this process for an e-mail.

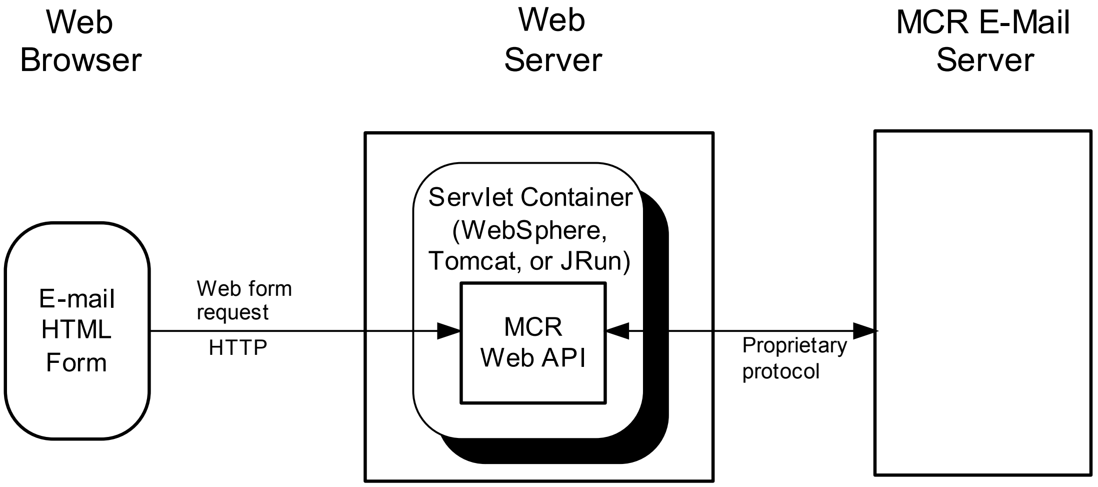
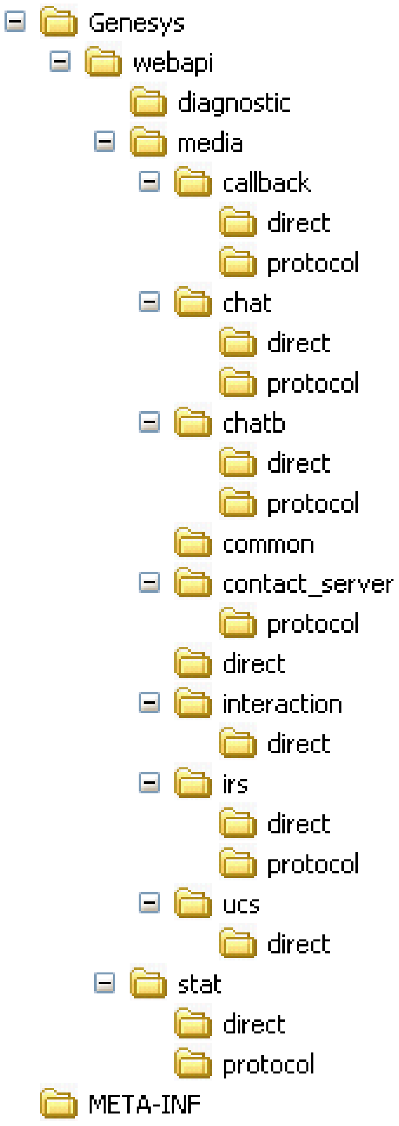


Figure 3: Sending an E-Mail to Multimedia (Container Could Also Be WebLogic)

Packages

The Web API's Java implementation consists of several packages that are organized by function and by media type (as shown in Figure 4 on page 24):

- `Genesys.webapi.media.common`—Classes used by all media types
- `Genesys.webapi.media.direct`—Contains the `direct` superclass
- `Genesys.webapi.media.interaction`—The Open Media package
- `Genesys.webapi.media.callback`—The voice callback packages
- `Genesys.webapi.media.chat`—The chat packages
- `Genesys.webapi.media.irs`—The e-mail packages
- `Genesys.webapi.media.ucs`—The Universal Contact Server packages
- `Genesys.webapi.stat`—The statistics packages



- Genesys
  - webapi
    - diagnostic
    - media
      - callback
        - direct
        - protocol
      - chat
        - direct
        - protocol
      - chatb
        - direct
        - protocol
      - common
      - contact_server
        - protocol
      - direct
      - interaction
        - direct
      - irs
        - direct
        - protocol
      - ucs
        - direct
    - stat
      - direct
      - protocol
  - META-INF

Figure 4: Web API Package Structure

Class Hierarchy

The non-statistics packages in the Web API are divided into two categories—the Open Media package (`Genesys.webapi.media.interaction`), which is provided only for Java in this release, and all the rest. If you are developing an Open Media application in Java, you will work primarily with the `_interaction_direct` class, which is in `Genesys.webapi.media.interaction`, and with the `_communication_exception` class, which is in `Genesys.webapi.media.common`.

To be comfortable working with the other packages in the Web API, you must understand the role of the two base packages: `Genesys.webapi.media.common` and `Genesys.webapi.media.direct`. The voice callback, chat, web collaboration, and e-mail packages extend classes that are defined in these two packages.

Genesys.webapi.media.common

This package contains 10 classes. Three of these classes are central to the workings of the Web API. Each one is the base class of an important object hierarchy:

- `_protocol_element`—A Packet object. See "Packet" on page 26 for more information.
- `_envelope`—Holds a packet. See "Envelope" on page 28 for more information
- `_envelope_factory`—Generates Envelope objects. See "Envelope Factory" on page 28 for more information.

The following classes are utility classes used for exception and data handling:

- `_communication_exception`—Handles communication exceptions.
- `_kvitem`—Represents data as a key-value pair.
- `_kvlist`—A list of `kvitem` objects.
- `_parse_exception`—Handles XML-parsing exceptions.

The Web API chat and e-mail packages each contain a `protocol` subpackage that is used to extend the `_protocol_element`, `_envelope`, and `_envelope_factory` classes. These subpackages are discussed in "Media Packages (Java)" on page 28.

Genesys.webapi.media.direct

This package contains only the `_direct_access` class. This `_direct_access` class extends the Java `Runnable` interface to make threading available to the Web API. The `connect()` method makes a socket connection to a media server, using host and port information. Your client application can either pass this information when it is instantiated, or pass it by means of the `connect()` method. The instance runs as a thread, and either listens to the incoming server data stream or sends client data to the server.

Clients can use the `_direct_access` class to contact any server that is compliant with the XML-envelope protocol schema. All subclasses that extend `_direct_access` must override the following methods:

```
public abstract void process(_protocol_element element);
public abstract void closed();
```

"Media Packages (Java)" on page 28 explains these two methods in detail.

The chat and e-mail packages each contain a `direct` subpackage, which in turn contains a class that extends the `_direct_access` class. These classes are called `_chat_direct` and `_irs_direct`, respectively.

Packets and Envelopes

Open 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. This concept is illustrated in Figure 5.

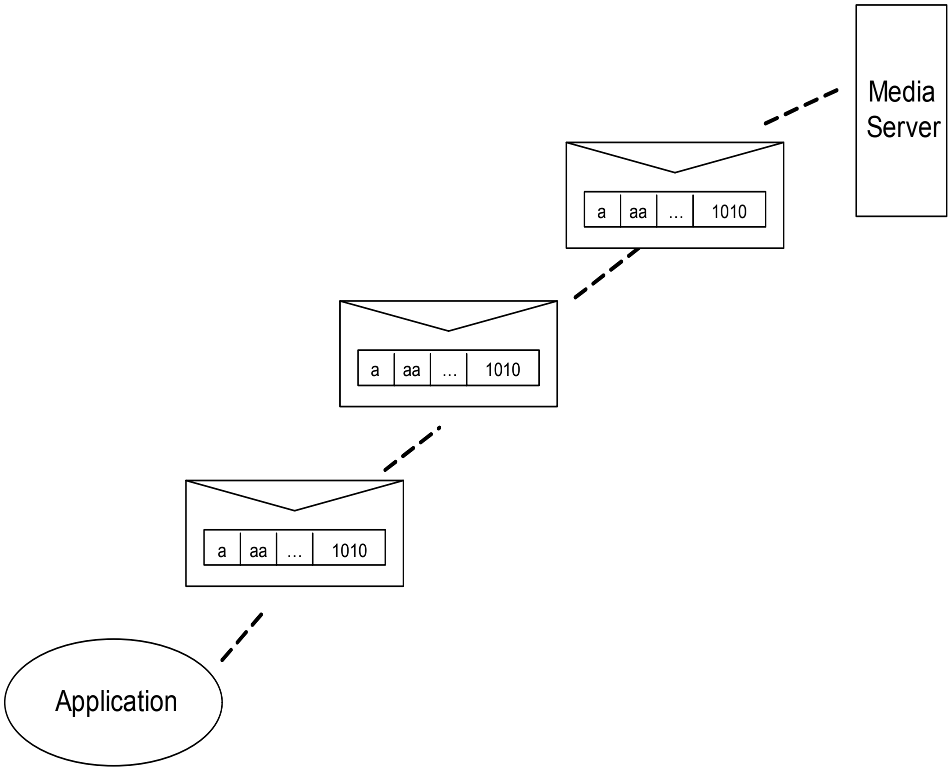


Figure 5: Packets and Envelopes

Packet

The root class of the packet hierarchy, `_protocol_element`, is an abstract class. The chat and e-mail packages each contain an abstract subclass of `_protocol_element`, respectively called `c_chat_packet` and `_irs_packet`. These subclasses define the constants used for their media types. The remaining classes in each media package extend the corresponding packet superclass. For example, _chat_request extends _chat_packet, which in turn extends _protocol_element. Figures 6 shows a diagram of this class relationship.

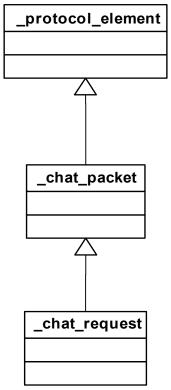


Figure 6: Sample Packet Hierarchy

The final child class (such as _chat_request in Figure 6 above) references the constants inherited from its direct superclass, and implements the interface methods inherited from the root packet class (_protocol_element).

Important Methods

_protocol_element defines two important methods that are implemented by its subclasses. These methods process the contents of the packet.

Pack Method

The pack(Document, Element) method adds media-specific elements or attributes into the <body> element, and may add kvlist objects. For example, the XML data may look something like the following in _chat_Request:

```
<envelope>
   ...
  <body>
   <requestJoin refId="xxx" userId="xxx" secureKey="xxx"
     queueKey="xxx" subject="xxx">
   </requestJoin>
  </body>
</envelope>
```

Parse Method

Every subclass overrides the parse() method inherited from the superclass. The (XML) Element object contains many attributes. If the subclass method finds the specific attributes that it needs, it extracts the corresponding data from the object.

Envelope

_envelope is the root class of the envelope class hierarchy. Each media package contains a class that extends the _envelope class. The subclass usually overrides only the parse() and pack() methods.

The pack() method in the _envelope class constructs a basic XML structure using the XML Document Object Model (DOM) API:

```
<envelope>
   <header>
      <protocol-info protocol-id="xxx", version-minor="xxx",
         version-major="xxx">
      </protocol>
  </header>
  <body>
  </body>
</envelope>
```

This structure is then passed into the pack() method in the subclass of the _protocol_element class. This, in turn, adds more information to the body element.

Envelope Factory

The _envelope_factory class is the root class of the envelope factory class hierarchy. Each media package has a class that extends _envelope_factory. The subclass does not implement any extra logic. It invokes the superclass methods.

Media Packages (Java)

This section discusses the Java implementation's media packages and their functionality.

Callback

Genesys.webapi.media.callback.protocol

Most of the classes in this package represent callback message packets, as shown in Figure 7 on page 29.

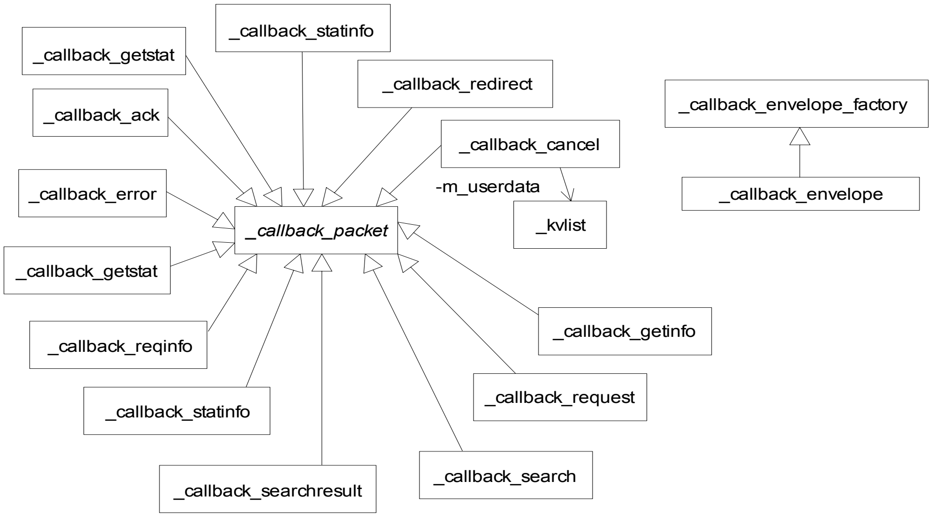


Figure 7: Genesys.webapi.media.callback.protocol Class Diagram

Dependencies

The `Genesys.webapi.media.callback.protocol` package uses classes from `Genesys.webapi.media.common`.

Classes

- `_callback_ack`—Encodes and decodes a callback packet containing an acknowledgement.
- `_callback_cancel`—Encodes and decodes packets containing requests to cancel a callback request.
- `_callback_envelope`—Can contain one or more callback protocol packets for sending to an instance of Callback Server.
- `_callback_envelope_factory`—A factory that creates `_callback_envelope` objects that can be sent to and received from a Callback Server.
- `_callback_error`—Encodes and decodes a packet containing a callback error.
- `_callback_getinfo`—Creates requests for callback information from the Callback Server.
- `_callback_getstat`—Represents a statistics request packet.
- `_callback_packet`—The abstract base class for all callback protocol packet classes.
- `_callback_redirect`—Redirects a packet to another Callback Server.
- `_callback_reqinfo`—Creates a callback request for information packet.
- `_callback_request`—Represents a callback request packet.
- `_callback_search`—Represents a search request.

- _callback_searchresult—Creates a search results packet that may contain information on one or more callback requests.
- _callback_statinfo—Represents a statistics information packet.

Genesys.webapi.media.callback.direct

Dependencies

The Genesys.webapi.media.callback.direct package uses classes in these packages:

- Genesys.webapi.media.common
- Genesys.webapi.media.direct
- Genesys.webapi.media.callback.protocol

Classes

This package only contains a single class, _callback_direct, which allows direct access to Universal Callback Server. This class extends the Runnable root class _direct_access, which makes the _callback_direct class a thread. Therefore, all the public method signatures in this class contain the synchronize keyword.

The class performs these operations:

- "Connect to Universal Callback Server"
- "Work with Universal Callback Server Data"
- "Handle Errors"

Connect to Universal Callback Server

The no-argument constructor instantiates a new object. You must use the connect() method afterwards to establish a connection with an instance of Universal Callback Server.

The constructor that takes arguments instantiates a new object and attempts to connect with the specified instance of Universal Callback Server.

Work with Universal Callback Server Data

The submit() method attempts to submit the packet to the specified instance of Universal Callback Server.

The reqid() method returns the request ID set for this instance.

The rc() method returns the result code after an operation. Universal Callback Server returns one of the values listed in Table 1 on page 31.

Handle Errors

The lasterror() method returns the last error. Universal Callback Server returns one of the values listed in Table 1 on page 31 or the default value.

Table 1: Return Codes for Universal Callback Server

Value	Error Code	Description
0	__rc_ok	Universal Callback Server accepts the request.
1	__rc_timeout	No response from Universal Callback Server.
2	__rc_closed	Lost or unestablished connection to Universal Callback Server.
3	__rc_failed	Request failed. Use the `lasterror()` method to get an error description.
4	__rc_internal	Internal error. Use the `lasterror()` method to get an error description.
5	__rc_confail	Connection failure.

Chat

The Chat Sample discussed in this guide uses the Flex Chat API to communicate with the Chat Server. The API enables direct access to a Chat Server, using the HTTP chat protocol to connect to and disconnect from the Chat Server and to establish and close chat sessions.

Genesys.webapi.media.chat.protocol

Most of the classes in this package represents e-mail message packets, as shown in Figure 8.

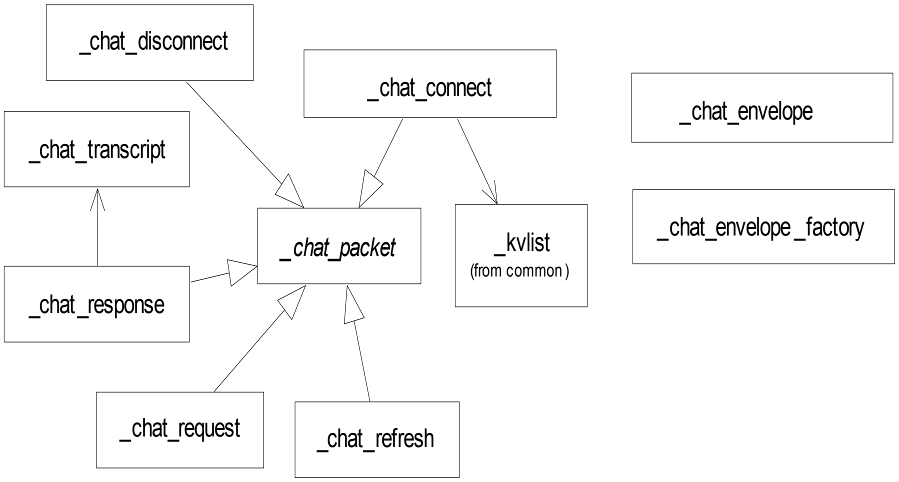


Figure 8: Genesys.webapi.media.chat.protocol Class Diagram

Dependencies

The protocol package uses classes in Genesys.webapi.media.common.

Classes

Eight classes are in this package. Here is a list of the classes. (See the *Multimedia 7.6 Web API Reference* or the *Web Media Platform SDK 7.6 .NET API References* for details in using these classes.)

- _chat_connect—This class connects a user to Chat Server.
- _chat_disconnect—This class encodes and decodes a request to close a chat session with a Chat Server.
- _chat_envelope—This class wraps a chat packet.
- _chat_envelope_factory—This class creates Envelope objects.
- _chat_packet—This class represents a chat message.
- _chat_response—This class encodes and decodes a Chat Server response to a request.
- _chat_request—This class encodes and decodes a request to join or create a chat session.
- _chat_transcript—This class encodes and decodes a chat session transcript.
- _chat_refresh—This class encodes and decodes a packet containing data refresh requests.

Genesys.webapi.media.chat.direct

The Genesys.webapi.media.chat.direct package contains the _chat_direct class, which client applications use to interact with Chat Server. This is the only class in this package.

The Chat Sample that is discussed in this book imports the package and uses the _chat_direct subclass of _direct_access. A client connects to, joins, and disconnects from a chat session using this class.

Dependencies

The Genesys.webapi.media.chat.direct package uses classes in these packages:

- Genesys.webapi.media.common
- Genesys.webapi.media.direct
- Genesys.webapi.media.chat.protocol

Classes

This package has only one class, _chat_direct. This class allows direct access to a Chat Server using the HTTP chat protocol. It extends the Runnable root class `_direct_access`, which makes the `_chat_direct` class a thread. Therefore, all the public method signatures in this class contain the `synchronize` keyword.

The class performs these operations:

- "Log In To Chat Server"
- "Log Out from Chat Server"
- "Create a Session"
- "Handle Errors"
- "Work with Chat Server Data"
- "Process Events"

Log In To Chat Server

A client application logs in and registers a user to a Chat Server via the `login()` method of `_chat_direct`. Users can identify themselves using nicknames. All user data provided with the request must contain certain keys that will be used for user identification and routing.

Log Out from Chat Server

To log out of a chat session using the `logout()` method, a client application must pass in the parameters `user_id and secure_key` from the last reply of the Chat Server.

Create a Session

One can either create a new web chat session, or join an existing chat session. For this second purpose, use the `join()` method with a different set of parameters.

Note: The web sample client does not include the ability to join an existing chat session. However, this ability can be provided by using the `join()` method and including the `session_id` parameter.

Handle Errors

Deal with errors using these methods:

- The `rc()` method returns the error code for the last-performed operation as reported by Chat Server.
- The `error()` and `errdesc()` methods return the error code and description, respectively, returned by Chat Server.
- The `lasterror()` method returns an error description for the last operation performed. Table 2 lists the possible error codes.

Table 2: Chat Server Error Code and Description

Error Code	Description
0	Success
1	Timeout occurred
2	Connection closed
3	Operation failed

Table 2: Chat Server Error Code and Description (Continued)

Error Code	Description
4	Internal failure
5	Connection failed
default	Unknown error

Work with Chat Server Data

Your client application must work with Chat Server at a relatively low level. Your application must maintain secure keys for transmitting transcript data, must track which users join and leave a session, and must frequently calculate the correct position for all the transcripts. There are many methods that help your application work with Chat Server:

- The `refresh()` methods send an update request to a Chat Server for the specified chat session.
- The `status()` method returns the status of the last operation, as reported by Chat Server.
- The `script_pos()` method returns the relative position of the chat transcript. The position value is where the client application has last received a transcript. Your application must supply this value, incremented by 1, in a subsequent refresh request. This is necessary so that Chat Server will send transcript updates starting from the specified transcript position. If this method returns string value `0,` then the chat session is over.
- The `secure_key()` method returns a key generated by Chat Server for security purposes. Your application must supply this value in subsequent requests to Chat Server.
- The `user_id()` method returns a user ID generated by Chat Server for identification purposes. Your application must supply this value in subsequent requests to Chat Server.
- The `timestamp()` method returns a timestamp from Chat Server in response to a refresh request.
- The `transcript()` method returns part of a chat transcript sent by Chat Server in response to a refresh request. This value can be `null` if there were no updates since the last refresh request. Otherwise, the transcript will contain a vector of classes of type `_event,` which describes events that took place in a chat session since the client's last refresh request.

Process Events

After a client application receives a `user connected` event, the first chat message may come from an agent application. Therefore, client applications should call `event.event_body()` to verify the presence of a chat message, in case of an error. Here is a code snippet showing how the application should check for this possibility:

```
if(event.event_type().equals
   (_chat_packet.__attrv_event_type_connect)){

   text2append = text2append + "New party ('"+ event.user_nick()
      + "') has joined the session";

   if (event.event_body() != null)
      text2append = text2append + ": " + event.event_body();
}
```

Log Out Gracefully 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.

E-Mail

Genesys.webapi.media.irs.protocol

Most of the classes in this package represent e-mail message packets, as shown in Figure 9.

Dependencies

The `Genesys.webapi.media.irs.protocol` package uses classes from `Genesys.webapi.media.common.`

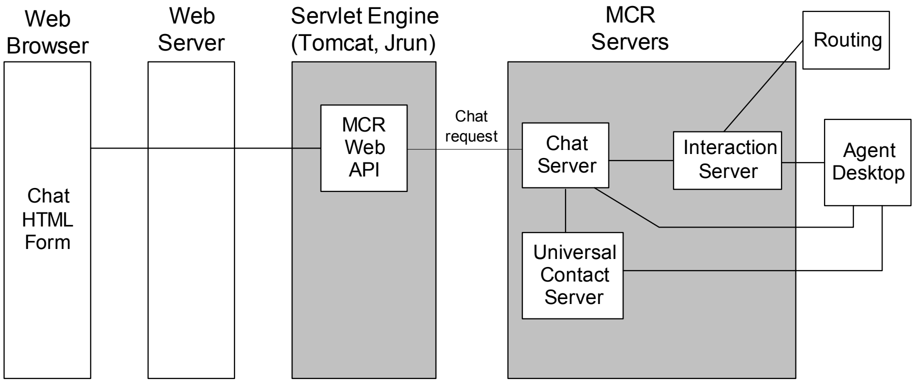


Figure 9: Genesys.webapi.media.irs.protocol Class Diagram

Classes

- `_irs_ack`—This class represents a protocol acknowledgment packet.
- `_irs_envelope`—This class constructs an e-mail protocol packet for E-mail Server Java and decodes information received from the server.
- `_irs_envelope_factory`—This class creates instances of e-mail envelopes for direct access to E-mail Server Java.

- _irs_error—This class represents an e-mail protocol error packet.
- _irs_getstat—Deprecated. Statistics functions are now carried out by the Genesys.webapi.stat packages. See "Statistics Packages" on page 39.
- _irs_getvrpstat—Deprecated. Statistics functions are now carried out by the Genesys.webapi.stat packages. See "Statistics Packages" on page 39.
- _irs_packet—This abstract class represents an e-mail packet.
- _irs_statinfo—Deprecated. Statistics functions are now carried out by the Genesys.webapi.stat packages. See "Statistics Packages" on page 39.
- _irs_submit—This class represents a packet containing a request to submit an e-mail.
- _irs_vrpstatinfo—Deprecated. Statistics functions are now carried out by the Genesys.webapi.stat packages. See "Statistics Packages" on page 39.

Genesys.webapi.media.irs.direct

Dependencies

The Genesys.webapi.media.irs.direct package uses classes in these packages:

- Genesys.webapi.media.common
- Genesys.webapi.media.direct
- Genesys.webapi.media.irs.protocol

Classes

This package only contains a single class, _irs_direct, which allows direct access to E-mail Server Java. This class extends the Runnable root class _direct_access, which makes the _irs_direct class a thread. Therefore, all the public method signatures in this class contain the synchronize keyword.

The class performs these operations:

- "Connect to E-Mail Server Java"
- "Work with E-Mail Server Java Data"
- "Handle Errors"

Connect to E-Mail Server Java

The no-argument constructor instantiates a new object. You must use the connect() method afterwards to establish a connection with an instance of E-mail Server Java.

The constructor that takes arguments instantiates a new object and attempts to connect with the specified instance of E-mail Server Java.

Work with E-Mail Server Java Data

The submit() method attempts to submit the packet to the specified instance of E-mail Server Java.

The reqid() method returns the request ID set for this instance.

The rc() method returns the result code after an operation. E-mail Server Java returns one of the values listed in Table 3 on page 37.

Handle Errors The lasterror() method returns the last error. E-mail Server Java returns one of the values listed in Table 3 on page 37 or the default value.

Table 3: Return Codes for E-Mail Server Java

Value	Error Code	Description
0	__rc_ok	E-mail Server Java accepts the request.
1	__rc_timeout	No response from E-mail Server Java.
2	__rc_closed	Lost or unestablished connection to E-mail Server Java.
3	__rc_failed	Request failed. Use the lasterror() method to get an error description.
4	__rc_internal	Internal error. Use the lasterror() method to get an error description.
5	__rc_confail	Connection failure.

Open Media

Genesys.webapi.media.interaction.direct

Dependencies

The Genesys.webapi.media.interaction.direct package uses the _communication_exception class in this package:

- Genesys.webapi.media.common

Classes

This package only contains a single class, _interaction_direct, which allows direct access to Interaction Server. This class extends the Runnable abstract class IPConnection, which allows _interaction_direct to run as a thread. Therefore, most of the public method signatures in this class contain the synchronize keyword.

The class performs these operations:

- "Connect to Interaction Server"
- "Work with Interaction Server Data"
- "Handle Errors"

Connect to Interaction Server The no-argument constructor instantiates a new object. You must use the connect() method afterwards to establish a connection with an instance of Interaction Server.

The constructor that takes arguments instantiates a new object and attempts to connect with the specified instance of Interaction Server.

The register() method registers a user with an Interaction Server using a specific media type.

The close() method closes the connection to Interaction Server.

Work with Interaction Server Data

The submit() method attempts to submit a packet to the specified instance of Interaction Server.

The change_properties() method updates the attached data of an interaction.

The stop_processing() method cancels the specified interaction.

The run() method starts an _interaction_direct thread that reads from a specific media server.

The get_interaction_id() method gets the interaction ID from the last server response.

The send() method encodes a packet and sends it to an instance of Interaction Server.

The rc() method returns the result code after an operation. Interaction Server returns one of the values listed in Table 4.

Handle Errors

The lasterror() method returns the last error. Interaction Server returns one of the values listed in Table 4 or the default value.

Table 4: Return Codes for Interaction Server

Value	Error Code	Description
0	__rc_ok	Interaction Server accepts the request.
1	__rc_timeout	No response from Interaction Server.
2	__rc_closed	Lost or unestablished connection to Interaction Server.
3	__rc_failed	Request failed. Use the lasterror() method to get an error description.
4	__rc_internal	Internal error. Use the lasterror() method to get an error description.
5	__rc_confail	Connection failure.

Universal Contact Server

Genesys.webapi.media.ucs.direct

Dependencies

The `Genesys.webapi.media.ucs.direct` package uses the `_communication_exception` class in this package:

- `Genesys.webapi.media.common`

Classes

- `_ucs_direct`—Retrieves or creates a contact and allows access to the contact's data.
- `_ucs_attribute`—Creates an attribute and allows you to set and retrieve its value or its hash code value. It is also used to determine if an attribute has been modified or lets you compare it to another object.
- `_ucs_interaction`—Creates an interaction and allows you access to the interaction's data, ID, and attributes.
- `_ucs_parameter_map`—Creates a parameter map and allows you to manipulate it.

Statistics Packages

The Web API lets you retrieve selected chat and e-mail statistics using the `stat_direct` class contained in package `Genesys.webapi.stat.direct`. The `getQueueStat` method of `stat_direct` retrieves the following statistics:

- `_stat_chat_total_distribution_time`—Average time lapse between the creation of a chat session and when the first meaningful response was sent.
- `_stat_chat_queue_length`—Total number of chat sessions that have been submitted and are awaiting processing.
- `_stat_chat_total_destributed`—Total number of chat sessions that have been submitted and processed.
- `_stat_webform_total_distribution_time`—Average time lapse between the creation of an e-mail and when the first meaningful response was sent.
- `_stat_webform_queue_length`—Total number of e-mails that have been submitted and are awaiting processing.
- `_stat_webform_total_destributed`—Total number of e-mails that have been submitted and processed.

The Chat-with-Statistics Sample discussed in this guide uses the statistics packages to gather the first three statistics. It then uses _stat_chat_total_distribution_time and _stat_chat_total_destributed to calculate the estimated wait time.

The Web API has two statistics packages, Genesys.webapi.stat.direct and Genesys.webapi.stat.protocol. Genesys.webapi.stat.protocol is not used directly by your applications, and will not be described further in this document.

Genesys.webapi.stat.direct

Dependencies

The Genesys.webapi.stat.direct package uses classes in these packages:

- com.genesyslab.statistics.lib
- Genesys.webapi.media.stat.protocol
- Genesys.webapi.media.common
- Genesys.ComLib.TKV
- Genesys.ML.Log

Classes

This package contains one class, stat_direct, which has a getQueueStat() method that is used to retrieve selected chat and e-mail statistics.

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 Compound and Simple Samples will not work properly. The ApplicationConstant.jsp file retrieves these options.

Web Sample and Configuration Access

The sample uses the ApplicationConstant.jsp page to retrieve options from the Multimedia Compound Sample Application object in Configuration Server. Table 5 on page 41 shows the configuration options needed and their default value if the retrieval is not successful.

Table 5: Configuration Options That Web API Retrieves

Section	Option	Default Value
chat	chat-queue	chat inbound queue
chat	chat-stat-interval	"" (an empty `String`) or null
e-mail	email-queue	inbound queue
e-mail	e-mail-stat-interval	"" (an empty `String`) or null
miscellaneous	applets-code-base	/CodeBase761
miscellaneous	stat-refresh-interval	30
miscellaneous	tenant	"" (an empty `String`) or null

Test Tool Access

The Multimedia test tools include files that access the Configuration Server during their diagnostic tests—`ChatSelfTest.jsp` and `EmailSelfTest.jsp`. These files retrieve the options from the Chat Server and E-mail Server Java `Application` objects, respectively. Chapter 12 on page 263 discusses the test tools in more detail.

Load Balancing

The Web API lets you take advantage of Multimedia's load-balancing features. This section briefly describes these features and how to use them. For more-detailed information, refer to *Multimedia 7.6 User's Guide*.

Load balancing provides greater scalability and service availability by providing multiple instances of certain Multimedia servers. The redundancy 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 Web API Server instances.

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 45), network performance, and so on.

Note: More sophisticated third-party solutions are available to balance traffic among multiple Web API Server instances. You can use one of these alternatives rather than the load balancing API described in this section.

Required Configuration

The Multimedia Load-Balancing API can load-balance data traffic across multiple instances of Web API Server. To do this, you must:

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

You can use Application Clusters to simplify configuration, as described below in "Using Load-Balancing API for Web API Server Instances".

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

Using Load-Balancing API for Web API Server Instances

Using the load-balancing API is very simple, as shown in the samples:

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

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

2. Use the method `inqSrvcByType("CfgAppType.CFGServerType", strTenantName)` or some other `inqSrvcByType` method, to select a Web API Server instance.

3. Use the methods getSrvcHost() and getSrvcPort() to produce redirect instructions.
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 method inqSrvcByAlias (strAliasName).

Here is a step-by-step procedure for using Web API Server load balancing:

1. For the first request when you do not know the alias of the server:

```
SvcDispatcher svcDispatcher = new SvcDispatcher();
// it is possible to use previously created SvcDispatcher
     instance if it is in the same thread
svcDispatcher.inqSrvcByType(CfgAppType.CFGWebAPIServer, strTenant);
if (getErrorCode() != 0) {
  // The required service was not found so do something else!
}
String strHost = svcDispatcher.getSrvcHost();
String strPort = svcDispatcher.getSrvcPort();
String strAlias2Remember = svcDispatcher.getSrvcAlias();
```

2. For the second request when you know the alias of the server, and want to connect to it again:

```
SvcDispatcher svcDispatcher = new SvcDispatcher();
// it is possible to use previously created SvcDispatcher
     instance if it is in the same thread
svcDispatcher.inqSrvcByAlias(strAlias2Remember);
if (getErrorCode() != 0) {
  // The required service was not found so do something else!
}
String strHost = svcDispatcher.getSrvcHost();
String strPort = svcDispatcher.getSrvcPort();
```

Samples

These two samples demonstrate load balancing between instances of Web API Server:

- icc_start.jsp
- icc_start_client.jsp

Both samples use the Load-Balancing API to select the available Web API Server. The difference is that the first sample uses a server-side redirect to the selected Web API Server, while the second sample uses a client-side redirect.

Note: The host where Web API Server is running must have access to Configuration Server, Solution Control Server, and Message Server.

Balancing Multiple Chat Servers

Required Configuration

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

Using Load Balancing API for Chat Server Load Balancing

Since 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.

The operation sequence shown in the chat sample application is shown here:

1. For every user request, create an instance of the class `SvcDispatcher`.

   It is possible to use only one instance if it is thread-safe.

2. When the user starts the chat session:
   a. Call the method `inqSrvcByType(CfgAppType.CFGChatServer, strTenant)` to select the Chat Server for the session.
   b. Call the method `getSrvcAlias()` to obtain the selected server’s alias and store it in the application. The server alias is transferred to and from the client part of the web application.

3. For all subsequent requests as the chat session continues, call the method `inqSrvcByAlias(<chat server allias>)` to obtain the Chat Server by its stored alias.

4. For any request, use the methods `getSrvcHost()` and `getSrvcPort()` to obtain the Chat Server host and port.

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 the 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 developing a chat application using the Multimedia Load-Balancing API, you must ensure that all subsequent requests regarding 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 this using the `chatRefreshTimeout` variable in the `...\WebAPISamples761\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 this interval is too short, the Web API Servers and Chat Servers will have to bear an unnecessarily high load. A shorter interval may require more instances of Web API Server, as described in "Using Load-Balancing API for Web API Server Instances" on page 42.
- A good rule of thumb is that the shorter the update time interval, the more Web API Servers are needed.

Balancing Multiple Instances of E-Mail Server Java

Required Configuration

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

Using Load Balancing API for E-Mail Server Java Instances

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

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

The e-mail sample application performs the following operations:

1. Gathers information from the form submission.

2. Gets e-mail host and port from the Load-Balancing API. The operations in this step are identical with those in “Using Load-Balancing API for Web API Server Instances” on page 42.
3. Prepares data for the Multimedia Web API.
4. Submits the request via the Multimedia Web API.
5. Returns the RefID of the request, possibly with an error code and description.

International Language Support

Multimedia 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—i18nsupport—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 multi-byte character set (MBCS). Your customized server pages (JSPs or ASPXs) must use this character set when interacting 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 Web API Server’s Application object.

Likewise, the media servers send Unicode data. The servlet (and engine) must respond to the client in the appropriate character set, which in this multi-byte case is not Unicode. However, this can only occur if the server knows both the correct character set and the correct code page for this data, because the same combination of MBCS characters may 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.

Here is a code snippet showing how a client using a different natural language can communicate with the server. The client gets the correct response based on the charset 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 correctly before returning 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 a user’s first name 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, this line does not change anything.

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

API Usage in the Samples

The Simple and Compound Samples use various parts of the Multimedia Web API to accomplish their tasks.

Note: Use this section in conjunction with the *Multimedia Web API Reference* (Javadoc), or with the *Web Media Platform SDK 7.6 .NET API References* (.CHM). These are located on the Genesys Developer Documentation Library DVD. They are also installed with the software, at the 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 95, and Chapter 11 on page 241.

All the samples access the `Genesys.webapi.media.common` (page 25) and `Genesys.webapi.media.direct` (page 25) classes. Each sample uses one or more specific APIs. Table 6 lists their API usage. For information on these packages, see “Media Packages (Java)” on page 28.

Table 6: API Access by Samples

Sample	Web APIs Used
E-mail	Genesys.webapi.system.loadbalancing Genesys.webapi.media.irs.direct Genesys.webapi.media.irs.protocol Genesys.webapi.media.common Genesys.webapi.utils.i18n
E-mail with Attachment	Genesys.webapi.system.loadbalancing Genesys.webapi.media.irs.direct Genesys.webapi.media.irs.protocol Genesys.webapi.media.common Genesys.webapi.utils.i18n

Table 6: API Access by Samples (Continued)

Sample	Web APIs Used
E-mail with Statistics	Genesys.webapi.system.loadbalancing Genesys.webapi.media.irs.direct Genesys.webapi.media.irs.protocol Genesys.webapi.media.common Genesys.webapi.utils.i18n Genesys.webapi.media.stat.direct Genesys.webapi.media.stat.protocol
Chat	Genesys.webapi.system.loadbalancing Genesys.webapi.media.chat.direct Genesys.webapi.media.chat.protocol Genesys.webapi.media.common Genesys.webapi.utils.i18n
Chat with Statistics	Genesys.webapi.system.loadbalancing Genesys.webapi.media.stat.direct Genesys.webapi.media.stat.protocol Genesys.webapi.media.chat.direct Genesys.webapi.media.chat.protocol Genesys.webapi.media.common Genesys.webapi.utils.i18n
Callback	Genesys.webapi.system.loadbalancing Genesys.webapi.media.callback.direct Genesys.webapi.media.callback.protocol Genesys.webapi.media.common Genesys.webapi.utils.i18n
Open Media (Interaction Submit)	Genesys.webapi.system.loadbalancing Genesys.webapi.media.interaction.direct Genesys.webapi.media.common Genesys.webapi.utils.i18n Genesys.webapi.confserv
Co-Browse	Genesys.webapi.system.loadbalancing Genesys.webapi.utils.i18n

Table 6: API Access by Samples (Continued)

Sample	Web APIs Used
Co-Browse Dynamic Start Page	Genesys.webapi.system.loadbalancing Genesys.webapi.utils.i18n
Co-Browse Init Start Page	Genesys.webapi.system.loadbalancing Genesys.webapi.utils.i18n
Co-Browse Meet Me	Genesys.webapi.system.loadbalancing Genesys.webapi.utils.i18n
ChatAndCo-Browse	Genesys.webapi.system.loadbalancing Genesys.webapi.media.chat.direct Genesys.webapi.media.chat.protocol Genesys.webapi.media.common Genesys.webapi.utils.i18n
FAQ	Genesys.webapi.utils.i18n Genesys.webapi.media.faq.direct

Chapter

2 About the Samples

This chapter briefly describes the sample applications that come with the Multimedia Web API. It also outlines how to install them. Then it describes the files and directories 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

This section describes the sample applications.

Simple Samples

Each Simple Sample application demonstrates a particular Web API feature or feature combination. Table 7 lists these samples and briefly describes them.

The .NET implementation includes the E-mail, Chat, Chat with AJAX, Callback, Open Media, Stat Server, and Universal Contact Server Simple Samples. The Java implementation includes all the Simple Samples listed in Table 7.

Table 7: Multimedia Simple Samples

Sample	Description
E-mail	A sample that demonstrates how to send e-mail.
E-mail with Attachment	A sample that demonstrates how to send e-mail with an attachment.

Table 7: Multimedia Simple Samples (Continued)

Sample	Description
E-mail with Statistics	A sample that demonstrates how to submit an e-mail interaction to E-mail Server Java via a web form, and to also retrieve statistics from Stat Server about the number of interactions in a queue (queue length), distribution volume, distribution time, and estimated wait time.
Chat	A sample that demonstrates how to initiate a chat session and implement user typing functionality.
Chat with Statistics	A sample that demonstrates how to initiate a chat session, and to also retrieve statistics from Stat Server about the number of interactions in a queue (queue length), distribution volume, distribution time, and estimated wait time.
Chat and Co-Browse	A sample that demonstrates how to initiate a chat session and initiate co-browsing during that session.
Callback	A sample that demonstrates how to submit a voice callback request.
Open Media (Interaction Submit)[a]	A sample that demonstrates how to use Open Media features to create a custom interaction.
Co-Browse	A sample that demonstrates how to initiate a co-browse session.
Co-Browse Dynamic Start Page	A sample that demonstrates how to initiate a co-browse session with a dynamic start page.
Co-Browse Init Start Page	A sample that demonstrates how to initiate a co-browse session with a specified start page.
Co-Browse Meet Me	A sample that demonstrates how to initiate a co-browse “meet me” session.
FAQ	A sample that demonstrates FAQ functionality.

[a] The Open Media sample is also variously referred to as the Interaction Submit Sample, Interaction Server Sample, and Custom Web Form Submit Sample. Its literal subdirectory name is `Itx-Submit`. These all refer to the same sample.

Compound Sample

The Multimedia Compound Sample, provided only for Java in this release, is a simple web application based on code from the web samples listed above in Table 7. The Compound Sample shows how to incorporate several Multimedia features into a single web application. Chapter 11 on page 241 explains the Compound Sample in detail.

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 you need not explicitly install them.

Tools You Need Before Installation

For information on what you will need before installing the samples, refer to *Genesys Supported Operating Systems and Databases*.

Installation Process

For complete installation instructions, see the *Multimedia 7.6 Deployment Guide*'s chapter on "Multimedia Configuration and Installation in Windows."

Installation Testing

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

```
http://<web_server_hostname>:<web_server_port>/TestTool761/index.html
```

Directory Structure

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

Java

Installing the Java implementation produces the following directory structures.

Tomcat

- `<tomcat_home>\webapps\CompoundSample761\`
- `<tomcat_home>\webapps\WebAPI761\`
- `<tomcat_home>\webapps\WebAPISamples761\`
- `<tomcat_home>\webapps\TestTool761\`

where `<tomcat_home>` is the directory that contains Tomcat.

JRun

- `<jrun_home>\servers\default\CompoundSample761\`
- `<jrun_home>\servers\default\default-ear\`
- `<jrun_home>\servers\default\SERVER-INF\`
- `<jrun_home>\servers\default\TestTool761\`
- `<jrun_home>\servers\default\WebAPI761\`
- `<jrun_home>\servers\default\WebAPISamples761\`

where `<jrun_home>` is the directory that contains JRun.

.NET

Installing 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 WebAPIServer761 web application. Contains some system and support files.
- `<Target folder>\.NET Web API Server & Samples\<WebAPI_Server_App_name>\TestTool761\`
  - Test Tools web application directory.
- `<Target folder>\.NET Web API Server & Samples\<WebAPI_Server_App_name>\SimpleSamples761\`
  - Examples web application directory.

- `<Target folder>\.NET Web API Server & Samples\<WebAPI_Server_App_name>\SimpleSamples761\Callback\`
  - Callback example directory.
- `<Target folder>\.NET Web API Server & Samples\<WebAPI_Server_App_name>\SimpleSamples761\Chat\`
  - Chat example directory.
- `<Target folder>\.NET Web API Server & Samples\<WebAPI_Server_App_name>\SimpleSamples761\ChatAjax\`
  - Chat with AJAX example directory.
- `<Target folder>\.NET Web API Server & Samples\<WebAPI_Server_App_name>\SimpleSamples761\Email\`
  - E-mail example directory.
- `<Target folder>\.NET Web API Server & Samples\<WebAPI_Server_App_name>\SimpleSamples761\ItxSubmit\`
  - Open Media example directory.
- `<Target folder>\.NET Web API Server & Samples\<WebAPI_Server_App_name>\SimpleSamples761\Statistics\`
  - Stat Server example directory.
- `<Target folder>\.NET Web API Server & Samples\<WebAPI_Server_App_name>\SimpleSamples761\UCS\`
  - Universal Contact Server example directory.

Simple Sample Files

Each Simple 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
- Callback
- Open Media
- Co-Browse
- FAQ
- Stat Server
- Universal Contact Server

Chat Samples

Three samples demonstrate chat-related functionality: Chat, Chat with Statistics, and Chat and Co-Browse. In this release, only the basic Chat and Chat with AJAX samples are provided for the .NET implementation.

Chat

The 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`—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 107 for more information.

Chat with Statistics

The Chat with Statistics Sample (provided only for Java in this release) has five files, which demonstrate how to create a chat session and retrieve chat statistics:

- `blank.jsp`—Used as a placeholder in a frame.
- `ChatStatInfo.jsp`—Retrieves and displays statistics information for a chat session.
- `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 on the chat panel.

Refer to “Chat with Statistics Sample” on page 117 for more information on this sample.

Chat and Co-Browse

The Chat and Co-Browse Sample (provided only for Java in this release) demonstrates how to initiate a chat session and to initiate co-browsing during that session. Its file base is almost identical to those of the two samples on which it is based: Chat, and Co-Browse. For more information on this sample, see “Chat and Co-Browse Sample” on page 143.

Chat with AJAX

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

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

Refer to “Chat with AJAX Sample” on page 192 for more information.

E-Mail Samples

Three samples demonstrate e-mail functionality: E-mail, E-mail with Attachment, and E-mail with Statistics. In this release, the .NET implementation combines the features of both the E-mail and the E-mail with Attachment 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:

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

Refer to "E-Mail Sample" on page 120 for more information on this sample.

E-mail with Attachment

The E-mail with Attachment Sample demonstrates how to create an e-mail with an attachment included, 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-mails that include attachments.

Refer to "E-Mail with Attachment Sample" on page 124 for more information on this sample.

E-mail with Statistics

The E-mail with Statistics Sample (provided only for Java in this release) demonstrates how to submit interactions (via a web form) to E-mail Server Java, while also polling Stat Server to retrieve statistics about the number of interactions in a queue (queue length), distribution volume, and distribution time.

Callback Sample

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

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

Refer to "Callback Sample" on page 101 for more information on this sample.

Open Media Sample

One sample demonstrates Open Media functionality. This sample consists of either one or two files:

- ItxSubmit.jsp (Java) or ItxSubmit.aspx.cs and ItxSubmit.aspx (.NET)—Contain code snippets that use the Open Media portion of the Web API to submit a custom interaction.

Refer to “Open Media Sample” on page 162 for more information on this sample.

Co-Browse Samples

Five samples (provided only for Java in this release) demonstrate web collaboration related functionality: Co-Browse, Co-Browse with Dynamic Start Page, Co-Browse with Initial Start Page, Co-Browse with Meet Me, and Chat and Co-Browse.

Co-Browse

One sample (provided only for Java in this release) demonstrates basic Co-Browse functionality. It is based on the Co-Browsing Server API, and consists of nine files:

- blank.html—Initializes empty frames used by the API. This is a default page.
- hbapi.html—Supports the client-side API and the co-browse applet.
- hbmessage_to_var.html—A messaging file that provides messages from the co-browse frame to the web application frame.
- hbmessage_to_var.js—A messaging file that provides messages from the co-browse frame to the web application frame.
- hbmessaging.js—JavaScript file containing the API of Co-Browsing Server.
- hbmessagingform.html—Increases security by sending Agent login information as a HTTP POST request instead of a GET request.
- qstring.js—A JavaScript file containing a utility class.
- CoBrowse.htm—The main frameset of the co-browse sample.
- CoBrowseEventHandler.jsp—Co-Browse event-receiver frame for “meet me” sample.

Refer to “Co-Browse Samples Overview” on page 137 for more information on this sample.

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

Chat and Co-Browse

The Chat and Co-Browse Sample demonstrates how to create a chat session and use web collaboration features. It consists of 11 files:

- `blank.html`—Initializes empty frames used by the API. This is a default page.
- `ChatAndCoBrowse.htm`—The main frameset of the sample.
- `hbapi.html`—Supports the client-side API and the co-browse applet.
- `hbmessage_to_var.html`—A messaging file that provides messages from the co-browse frame to the web application frame.
- `hbmessage_to_var.js`—A messaging file that provides messages from the co-browse frame to the web application frame.
- `hbmessaging.js`—JavaScript file containing the API of Co-Browsing Server.
- `hbmessagingform.html`—Increases security by sending Agent login information as a HTTP `POST` request instead of a `GET` request.
- `qstring.js`—A JavaScript file containing a utility class.
- `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 on the chat panel and handles co-browse functionality.

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

Refer to "Chat and Co-Browse Sample" on page 143 for more information on this sample.

Co-Browse Dynamic Start Page

One sample demonstrates Co-Browse Dynamic Start Page functionality.
Its core logic resides in this file:

- `ExampleOfDynamicStartPage.jsp`

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

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

Refer to "Co-Browse Dynamic Start Page" on page 152 for more information on this sample.

Co-Browse Init Start Page

One sample demonstrates Co-Browse Init Start Page functionality. It consists of nine files:

- `blank.html`—Initializes empty frames used by the API. This is a default page.
- `CoBrowseEventHandler.jsp`—Cobrowse event-receiver frame.
- `hbapi.html`—Supports the client-side API and the co-browse applet.
- `hbmessage_to_var.html`—A messaging file that provides messages from the co-browse frame to the web application frame.
- `hbmessage_to_var.js`—A messaging file that provides messages from the co-browse frame to the web application frame.
- `hbmessaging.js`—JavaScript file containing the API of Co-Browsing Server.
- `hbmessagingform.html`—Increases security by sending Agent login information as a HTTP `POST` request instead of a `GET` request.
- `InitialStartPageExample.html`—Start a co-browser session from this page.
- `qstring.js`—A JavaScript file containing a utility class.

Refer to “Co-Browse Init Start Page” on page 150 for more information on this sample.

Co-Browse With Meet Me

One sample demonstrates Co-Browse with Meet Me functionality. It consists of nine files:

- `blank.html`—Initializes empty frames used by the API. This is a default page.
- `CoBrowse.htm`—The main frameset of the co-browse sample.
- `CoBrowseEventHandler.jsp`—Cobrowse event-receiver frame for “meet me” sample.
- `hbapi.html`—Supports the client-side API and the co-browse applet.
- `hbmessage_to_var.html`—A messaging file that provides messages from the co-browse frame to the web application frame.
- `hbmessage_to_var.js`—A messaging file that provides messages from the co-browse frame to the web application frame.
- `hbmessaging.js`—JavaScript file containing the API of Co-Browsing Server.
- `hbmessagingform.html`—Increases security by sending Agent login information as a HTTP `POST` request instead of a `GET` request.
- `qstring.js`—A JavaScript file containing a utility class.

Refer to “Co-Browse Meet Me” on page 148 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" on page 157 for more information on this sample.

Stat Server Sample for .NET

One sample demonstrates statistics functionality. It consists of two files:

- `StatInfo.aspx.cs` and `StatInfo.aspx`—Contain code snippets that retrieve statistics about interactions from Stat Server.

Refer to "Stat Server Sample" on page 217 for more information on this sample.

Universal Contact Server Sample for .NET

The Universal Contact Server sample has four files, which demonstrate e-mail history functionality:

- `Action.aspx.cs` and `Action.aspx`—Controls the behavior of the page.
- `UCS.aspx.cs` and `UCS.aspx`—Controls most of the page presentation to the user.

Refer to "Universal Contact Server Sample" on page 226 for more information.

Compound Sample Files

The Compound Sample is provided only for Java in this release. It includes some nine folders and 56 files, of which 25 are graphics files. The rest of the files are mostly from the Simple Samples. The files are organized into these categories:

- "Graphics"
- "Authentication"
- "Graphical User Interface"
- "Servlet Container"
- "Media"

Graphics

This folder contains image files used in the Compound Samples.

Images Folder

Contains `.gif` files used in the Compound Sample GUI display.

Authentication

These folders contain files that identify and authenticate users. The user information entered is also used as attached data in various media requests.

AccountInfo Folder

- `PersonalInfo.jsp`—Presents a form in which users can enter their personal information.

Login Folder

- `CheckLogin.jsp`—Validates a user’s login credentials.
- `Login.jsp`—Presents a login page for users.
- `Logout.jsp`—Allows users to log out of a session.

Graphical User Interface

This folder contains `.jsp` files used to create the user interface.

MainWindow Folder

- `AllMedia.jsp`—Determines which media are working and available for use in the web sample.
- `Command.jsp`—Presents the command tabs for the sample.
- `LeftNavFrame.jsp`—Allows users to navigate the sample website.
- `MainFrame.jsp`—Presents the main entry page for the sample.

Servlet Container

All web applications running in a servlet container—such as WebSphere, WebLogic, Tomcat, or JRun—are required to have these folders and files. You can leave them as they are, or edit them to include application-specific information for the servlet container.

WEB-INF Folder

- `web.xml`—Application-level configuration file for use by servlet container.

META-INF Folder

- `MANIFEST.MF`—Application-level manifest file for use by servlet container.

Media

The folders under this category are organized according to the media they represent. The files in these folders are mainly from the simple web samples.

Chat Folder

- blank.jsp—Used as a placeholder in a frame.
- ChatStatInfo.jsp—Retrieves and displays statistics information for a chat session.
- 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 on the chat panel.
- ChatOptions.jsp—Checks for chat media availability and presents a chat window only if chat is available.
- ChatTranscript.jsp—Supports popular Internet expressions such as smiling or frowning facial expressions and hyper links in an e-mail or chat communication.

Email Folder

- Email.jsp—Contains Java code snippets that use the e-mail portion of the Web API to send e-mails.
- EmailHistory.jsp—Gets customer interactions history from Universal Contact Server.
- EmailHistoryFrameset.jsp—The frameset for the e-mail history functionality.
- EmailOptions.jsp—Shows all available e-mail options like web form submission, e-mail history from Universal Contact Server.
- PrintHistory.jsp—Prints selected thread or single e-mail. Also helps to update attach data of the interaction.

Callback Folder

- HtmlCallback.jsp—Provides functionality similar to the Callback JSP used in the Simple Sample.
- CallbackOptions.jsp—Checks for callback availability and presents the callback JSP only if the callback service is available.
- calendar.jsp—Provides a JavaScript calendar window.

Help Folder

- AllMediaHelp.jsp—States that the user can access all media requests.
- ChatOptionsHelp.jsp—Explains to a user how to choose chat options.

- EmailHelp.jsp—Explains to users how to create an e-mail request from the sample.
- CallbackOptionsHelp.jsp—Explains to a user how to choose chat options.
- Help.jsp—Contains general help message for the sample. It is a simple FAQ example.
- HtmlChatPanelHelp.jsp—Explains to users how to use the HTML Chat Sample request.
- PersonalInfoHelp.jsp—Explains to users that they must fill in their personal information.

Test Tool Files

Java

The Java implementation provides these test tools:

- blank.html—Initializes empty frames used by the API. This is a default page.
- CallbackSelfTest.jsp—Validates Multimedia callback web API.
- ChatSelfTest.jsp—Validates Multimedia chat web API.
- ComboTestPage.jsp—Validates system load-balancing installation.
- EMailSelfTest.jsp—Validates Multimedia 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.

.NET

The .NET implementation provides these test tools:

- diagnostic.xsl—Stylesheet file for diagnostic routines.
- LBTest.aspx.cs—Partial class for LBTest.aspx page: validates system load-balancing installation.
- LBTest.aspx—Displays LBTest.aspx.cs.
- EmailSelfTest.aspx.cs—Partial class for EmailSelfTest.aspx page: validates Multimedia e-mail .NET API.
- EmailSelfTest.aspx—Displays EmailSelfTest.aspx.cs.
- ChatSelfTest.aspx.cs—Partial class for ChatSelfTest.aspx page: validates Multimedia chat.NET API.
- ChatSelfTest.aspx—Displays ChatSelfTest.aspx.cs.
- CallbackSelfTest.aspx.cs—Partial class for CallbackSelfTest.aspx page: validates Multimedia callback .NET API.

- CallbackSelfTest.aspx—Displays CallbackSelfTest.aspx.cs.
- StatSelfTest.aspx.cs—Partial class for StatSelfTest.aspx page: validates Multimedia stat server .NET API.
- StatSelfTest.aspx—Displays StatSelfTest.aspx.cs.
- UCSSelfTest.aspx.cs—Partial class for UCSSelfTest.aspx page: validates Multimedia Universal Contact Server .NET API.
- UCSSelfTest.aspx—Displays UCSSelfTest.aspx.cs.

Chapter

3 Understanding and Using the E-Mail Service

Multimedia’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:

To fully understand the e-mail service, consult the *Multimedia 7.6 Deployment Guide*, *Multimedia 7.6 User’s Guide*, *Universal Routing 7.6 Business Process User’s Guide*, *Universal Routing 7.6 Reference Manual*, and *Framework 7.6 Stat Server User’s Guide*.

Overview

Figure 10 shows the architecture and components involved in an e-mail service. The *Multimedia 7.6 Deployment Guide* explains the behavior of E-mail Server Java in greater detail.

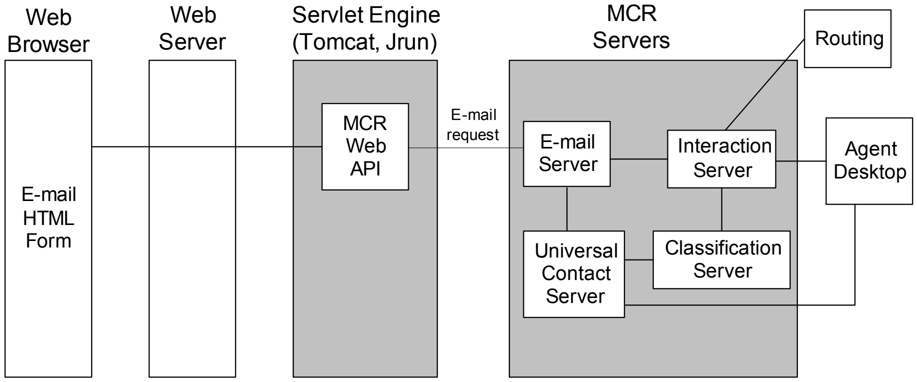


Figure 10: Web-Based E-Mail Service Architecture

Life Cycle of an E-Mail Session

The life cycle of an e-mail session is quite complicated. Figure 11 depicts this life cycle from the perspective of the Multimedia E-mail Sample. The initial state is the `empty` state. The only action available here is to connect to E-mail Server Java. 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.

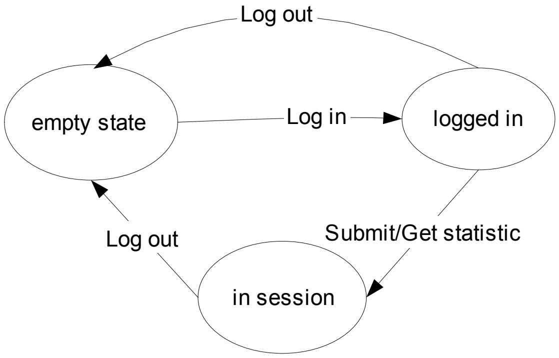


Figure 11: State Diagram of an E-Mail Session

Event Flow of an E-Mail Session

This section presents the event flow of an e-mail session, from a web-client perspective. Figure 12 shows the event flow from the Web sample to E-mail Server Java. The rightmost box represents the workflow-control components, such as knowledge management (classification), routing, and strategies, if any, as a single entity.

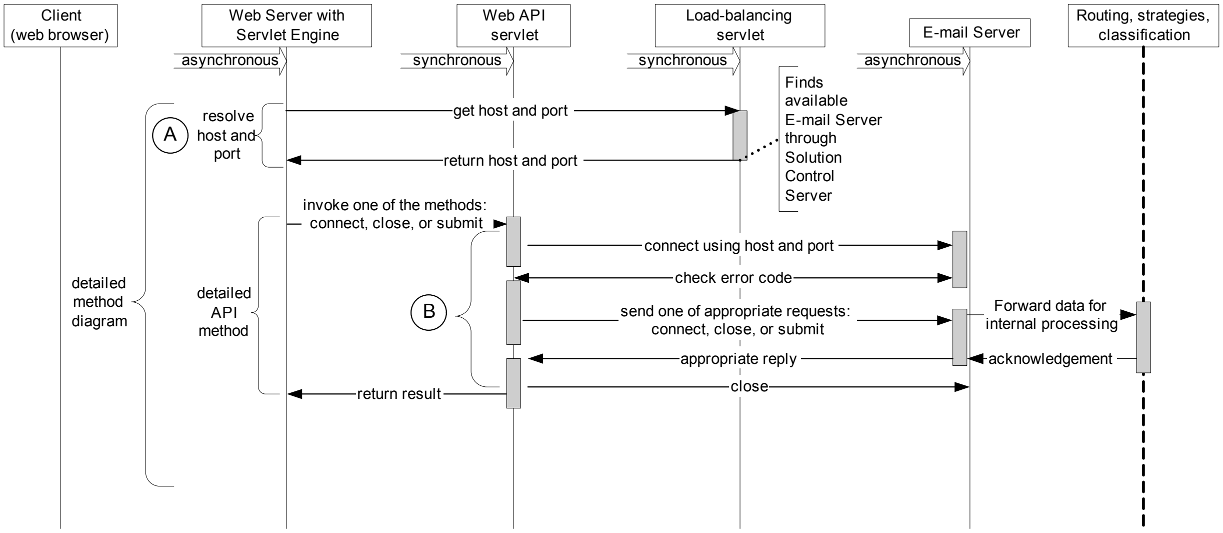


Figure 12: Event Flow Between the Web API Servlet and E-Mail Server Java

Note: Figure 12 shows a detailed view of the event flow between the Web API servlet and E-mail Server Java. It identifies certain sections of the event flow with the labels A and B. Figure 13 on page 70 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 Java. E-mail Server Java 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 Java. Any subsequent activity is not considered to be part of the e-mail architecture, but rather part of the Multimedia architecture.

Figure 13 on page 70 depicts the event flow of an HTTP request for connection, statistic retrieval, or submission.

For more information on how Multimedia processes an e-mail, refer to the *Multimedia 7.6 Deployment Guide.*

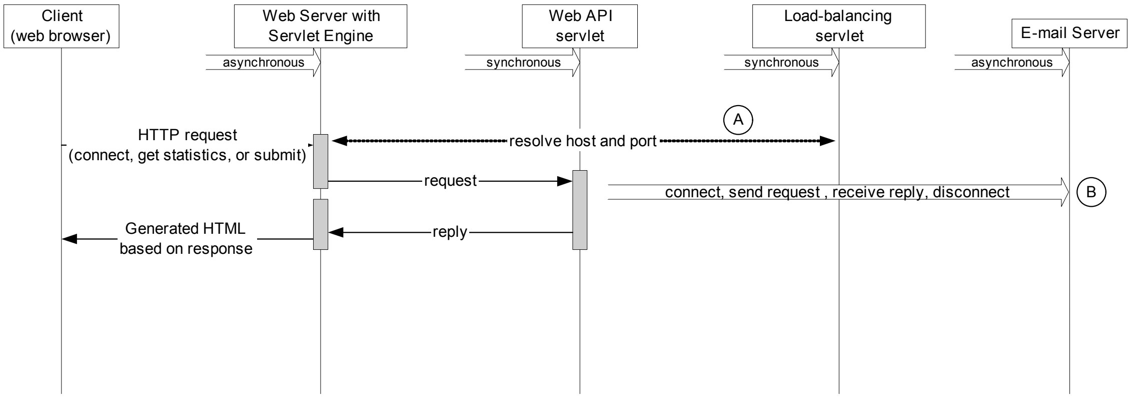


Figure 13: Event Flow for a Connection, Statistic Retrieval, or Submission Request

Chapter

4 Understanding and Using the Flex Chat Service

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

Overview

This section illustrates the concepts and architecture for the chat service. Figure 14 depicts the components involved in a chat service. This chapter only discusses the browser, web server, Web API, and Chat Server. The *Multimedia 7.6 Deployment Guide* discusses Chat Server in more detail.

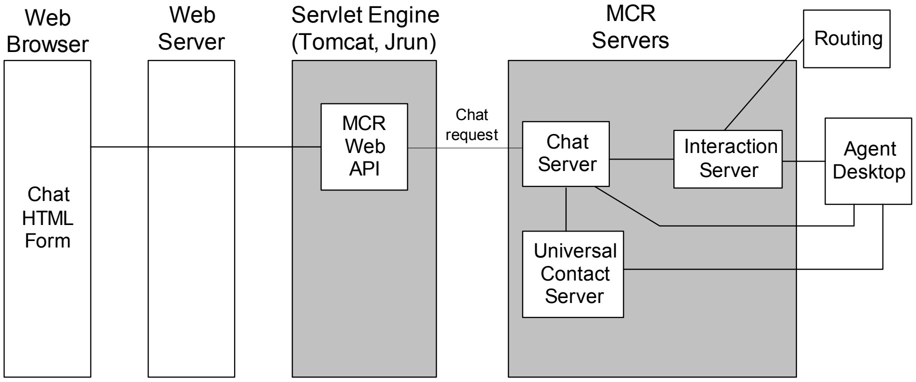


Figure 14: Architecture of the Flex Chat Service

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 15 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 Multimedia 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 *Multimedia 7.6 Web API Reference* or the *Web Media Platform SDK 7.6 .NET API References.*

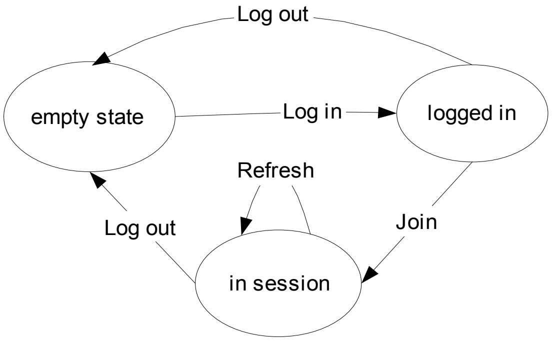


Figure 15: State Diagram of a Chat Session

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 75 for more information.

Event Flow of a Chat Session

This section details the event flow of a typical chat session (see Figure 16 on page 74).

In the first request in a chat session, the API servlet must request the alias for a Chat Server (flow A in Figure 16 on page 74). 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 72).

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 16 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 17 on page 74 and Figure 18 on page 75 abstract those sections of the diagram by referencing these labels.

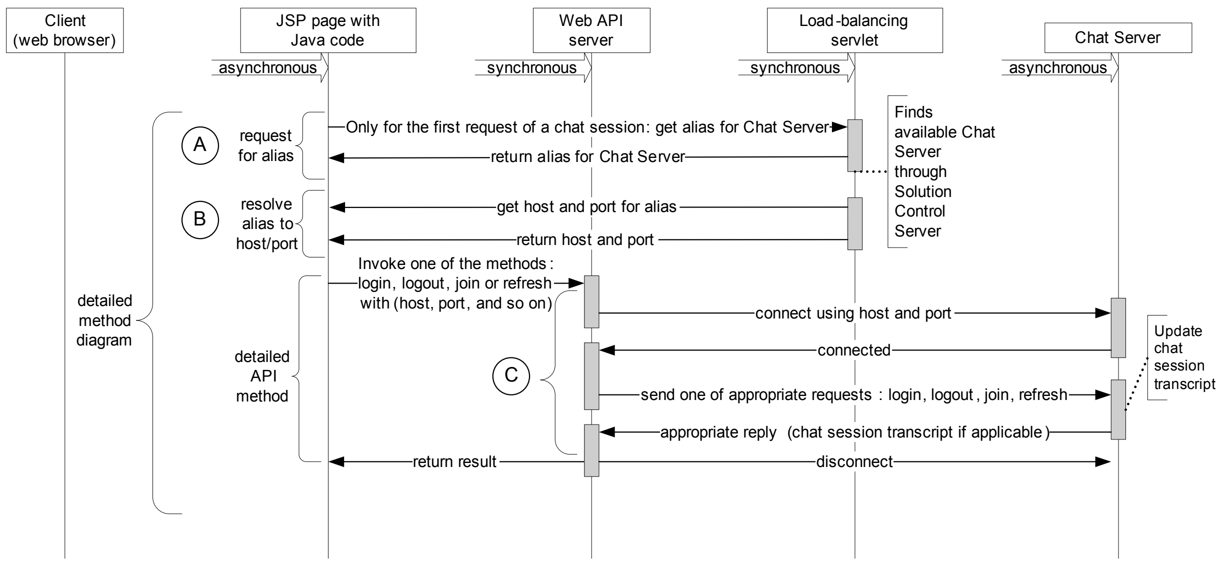

Client (web browser)
JSP page with Java code
Web API server
Load-balancing servlet
Chat Server
asynchronous
synchronous
synchronous
asynchronous
A
request for alias
Only for the first request of a chat session: get alias for Chat Server
return alias for Chat Server
Finds available Chat Server through Solution Control Server
B
resolve alias to host/port
get host and port for alias
return host and port
detailed method diagram
Invoke one of the methods: login, logout, join or refresh with (host, port, and so on)
connect using host and port
Update chat session transcript
detailed API method
C
connected
send one of appropriate requests : login, logout, join, refresh
appropriate reply (chat session transcript if applicable)
return result
disconnect

Figure 16: Request/Event Flow Between Multimedia Components for Chat Service

Figure 17 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 16.

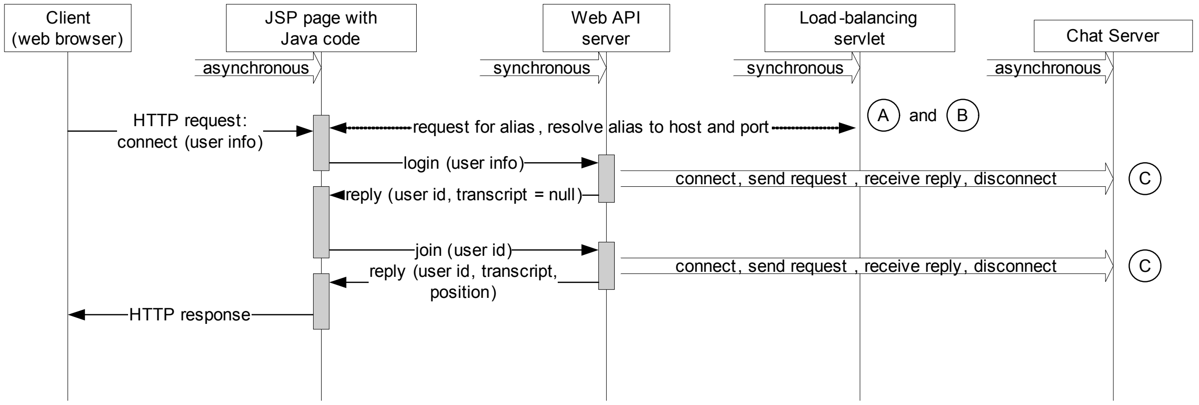

Client (web browser)
JSP page with Java code
Web API server
Load-balancing servlet
Chat Server
asynchronous
synchronous
synchronous
asynchronous
HTTP request: connect (user info)
request for alias, resolve alias to host and port
A
and
B
login (user info)
connect, send request , receive reply, disconnect
C
reply (user id, transcript = null)
join (user id)
reply (user id, transcript, position)
connect, send request , receive reply, disconnect
C
HTTP response

Figure 17: Event Flow for Connect Request

Figure 18 on page 75 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.

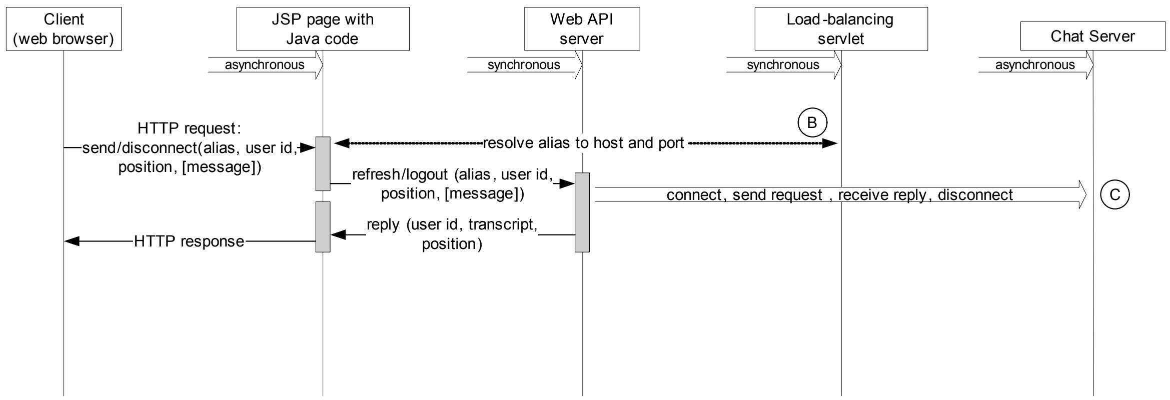


Figure 18: Event Flow for Send or Disconnect Request

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 (see “Chat” on page 31) is a vector of transcript items (objects). It contains members as described in “Transcript Description,” above.

The transcript() method of the _chat_direct class (see “Chat” on page 31) 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 Callback Service

This chapter describes the Multimedia voice callback service, in the following sections:

Overview

Genesys Voice Callback helps contact centers manage periods of high inbound call volume, by providing an additional channel for customer contact when the call load is heavy. This option allows callers to request a callback from an agent instead of waiting on hold. Voice Callback supports two types of callbacks:

- **ASAP** (as soon as possible) callback.
- **Scheduled** callback (for a specific date and time).

Multimedia's callback service allows you to write software that takes advantage of Voice Callback. This chapter explains the service's architecture and event flow. For more information on Voice Callback, consult *Voice Callback 7.6 Deployment Guide* and *Voice Callback 7.6 Reference Manual*.

Architecture

The Multimedia callback service relies on the Universal Callback Server to transmit callback requests to agents, as shown for web-submitted requests in Figure 19.

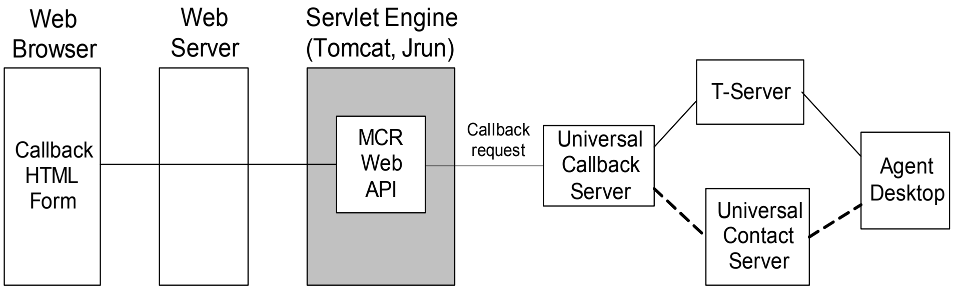


Figure 19: Web-Based Callback Service Architecture

Event Flow of a Callback Request

The upper part of Figure 20 on page 79 shows the event flow from the Web sample, running on a browser, to the Callback Server.

After the Web sample connects to the Callback Server, it submits a callback request. When the Callback Server receives the request, it optionally passes the request to the Contact Server. It then sends an acknowledgement to the Web sample. At this point, the sample disconnects from the Callback Server.

Note: Although Figure 20 shows interactions with a Contact Server, the Contact Server is optional and is not used for handling callback requests. Its only callback-related function is to receive copies of the callback interactions, so it can place them in the contact history.

The lower section of Figure 20 shows the basic interactions among the Callback Server, Contact Server, T-Server, Routing Server, and Agent Desktop, after the Callback Server has received and acknowledged a callback request from the Web sample.

First, the Callback Server sends the callback request to the Routing Server, in the form of a T-Server event with attached data. The Routing Server selects an appropriate agent and tells the Callback Server how to route the request. At this point, the Callback Server sends the callback request to an agent.

When the agent accepts, the Callback Server optionally notifies the Contact Server. When the agent has finished the callback, the Callback Server may also notify the Contact Server.

Other types of callback-related activity, such as a cancellation or a request for information about a callback request, have the same structure as the upper section of Figure 20.

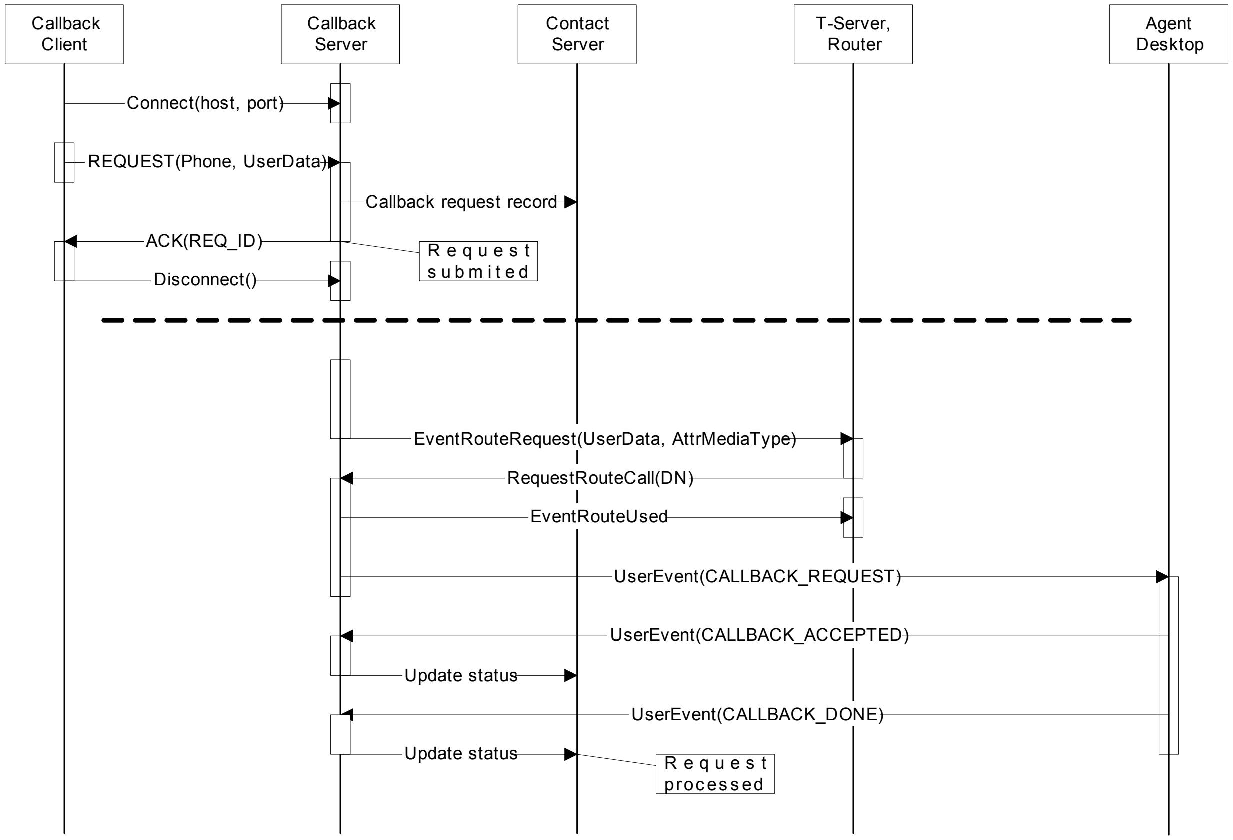


Figure 20: Event Flow Between the Callback Server and Other Components

Chapter

6 Understanding and Using the Open Media Service

This chapter discusses the Multimedia Open Media service, in the following sections:

Overview

Genesys Open 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, Open 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 Multimedia's built-in chat or e-mail types.

The Web API implementation of Open 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

Multimedia's implementation of Open Media relies on Interaction Server, as shown in Figure 21. 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.

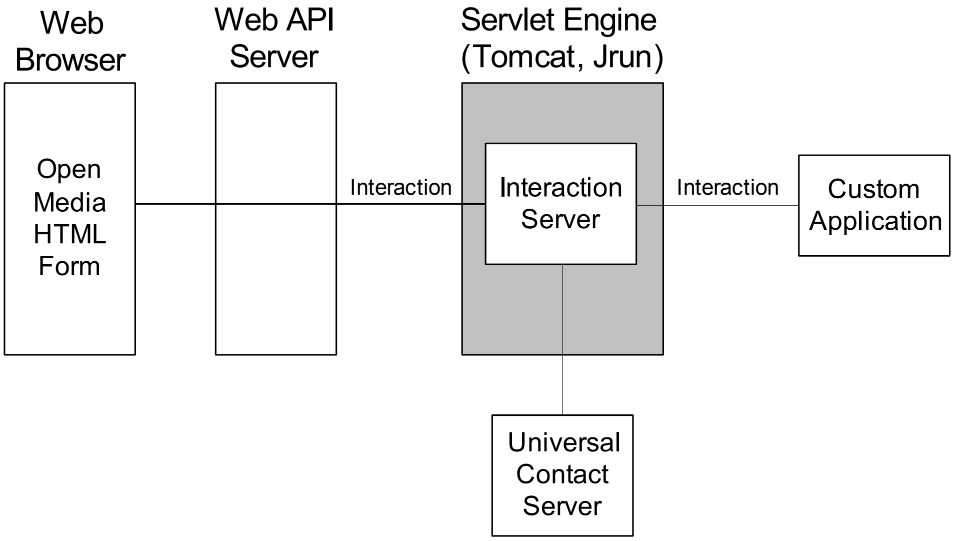


Figure 21: Web-Based Open Media Service Architecture

Event Flow of an Open Media Request

Figure 22 on page 83 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 22 on page 83.

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

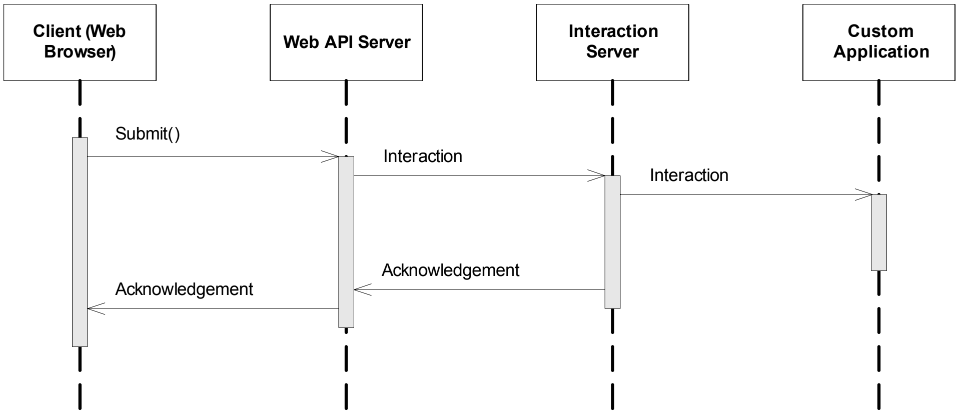


Figure 22: Event Flow Between the Interaction Server and Other Components

Chapter

7 Understanding and Using the Web Collaboration Service

This chapter explains the concept of web collaboration, and provides a high-level view of Co-Browsing Server's architecture. This server adds co-browse capabilities to Multimedia web-based clients.

Note: Co-Browse samples are provided only for Java in this release.

The chapter contains the following sections:

What is Co-Browsing?

Co-Browsing Server is a web-based application that enables agents and customers to co-browse 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 co-browsing the Yahoo! home page, when you click the `News` link, your co-browsing partner also sees the Yahoo! `News` page.

The actions that the participants in a co-browsing 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

Co-Browsing Server acts as a proxy between the clients and the site navigated. Co-Browsing Server sits between the clients and the servers at the website. Figure 23 shows Co-Browsing Server's high-level architecture.

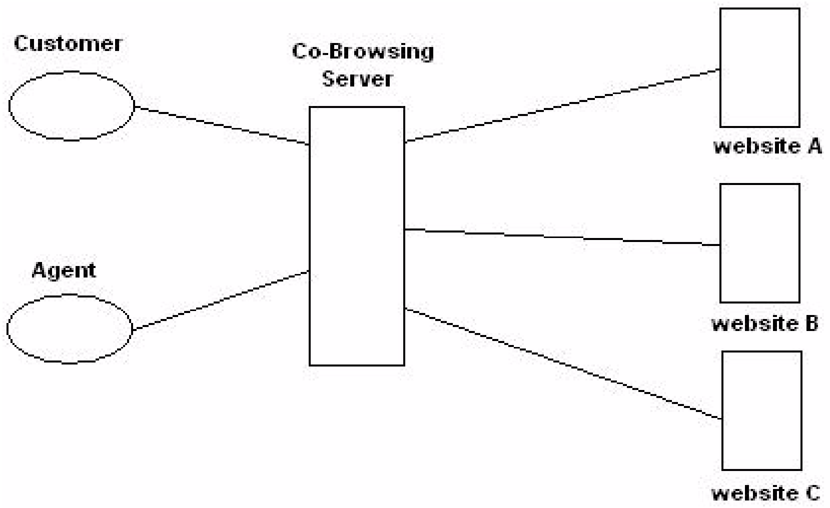


Figure 23: Co-Browsing Server Proxy Architecture

Web Collaboration Process

Figure 24 shows the typical sequence of steps in the web collaboration process between an agent and a customer.

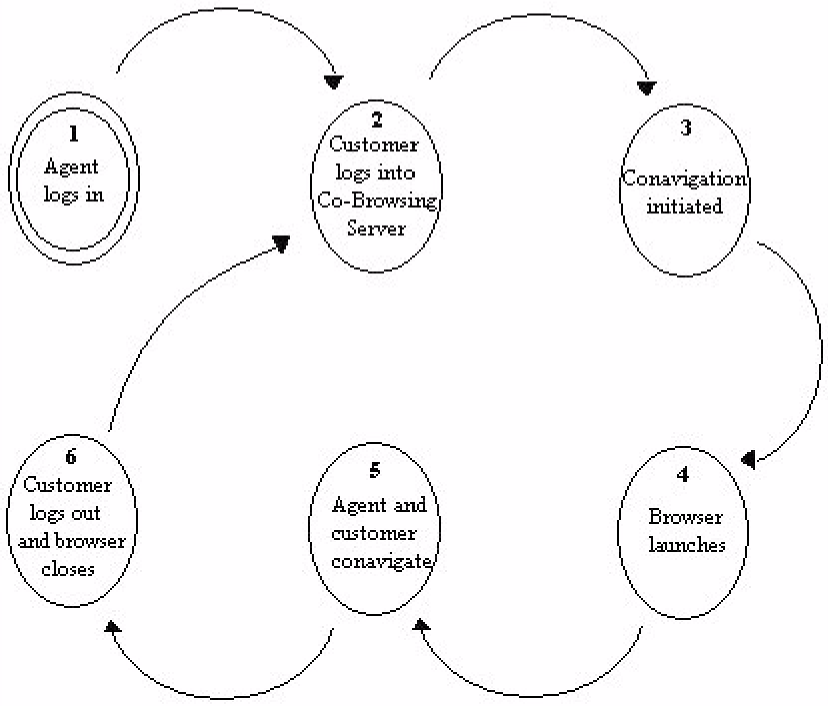

1
Agent logs in
2
Customer logs into Co-Browsing Server
3
Conavigation initiated
4
Browser launches
5
Agent and customer conavigate
6
Customer logs out and browser closes

Figure 24: Web Collaboration Process

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

Note: Co-Browsing 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 Co-Browsing Server API.

Integrating Co-Browsing into Your Application

In order to integrate co-browse into your client application, you must modify your application to include some co-browse 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 to implement callback functions, how to create initial and dynamic start 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 Co-Browsing Server’s API.
- `hbcallback.js`—JavaScript file containing default callback API implementations.
- `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 co-browse 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 co-browse frame to the web application frame.
- `hbmessage_to_var.js`—A messaging file that provides messages from the co-browse 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 Co-Browsing Server installation. The default installation puts them under the `<Co-browse server home>/clientapi/` directory.

- `toplevelframeset.html`—a frameset that holds the frames for callback and co-browse messaging files.
- `varExampleForm.html`—a simple example client that captures and sends events to Co-Browsing Server.
- `example_callbacks.html`—a sample implementation of client callbacks provided by the co-browse applet.

Design, Deployment, and Configuration Details

For co-browsing application design, deployment, and configuration guidelines, see the KANA Response Live documentation listed in "Related Resources" on page 17. (Genesys licenses certain co-browsing components from KANA Software, Inc.)

Creating a Co-Browse Session in a Separate Browser Window

The Co-Browsing Server client API supports two different ways of creating co-browse 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 Co-Browse sample that is described in "Co-Browse Samples" on page 58.

You cannot use the `HBInitEmbeddedFrame` function when you have a Co-browse 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

8 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

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 *Multimedia 7.6 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 n 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

9 Multimedia Simple Samples for Java

This chapter examines Genesys Multimedia's simple, web-based samples for Java, and their code. (.NET developers should instead see Chapter 10, "Multimedia Simple Samples for .NET," on page 169.) This chapter covers the following topics:

Overview

This chapter explains how to implement voice callback, chat, e-mail, web collaboration, Open Media, and FAQ functions in your web application through a review of the key functions in the respective samples. The samples come with the Multimedia Interactive Management CD. See Chapter 2, “About the Samples,” on page 51 for information on installing the samples.

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 *Multimedia 7.6 Web API Reference* (Javadoc) for further information.
- Although this chapter reviews the different functions that the sample performs, some of these functions and the excerpted code may not be presented in the same order or layout as the sample.

Also note that Chapter 11 includes sections on customizing code. It identifies, under the subheading “Customization Notes,” the areas in the Compound Sample where you can modify your application. These sections may also be helpful as you work with the Simple Samples.

Sample Overview

The Simple Samples discussed here are:

- Web-Based Voice Callback
- Web-Based Chat
- Web-Based Chat with Statistics
- Web-Based Chat and Co-Browse
- Web-Based E-mail
- Web-Based E-mail with Attachment
- Web-Based E-mail with Statistics
- Web-Based Open Media
- Web-Based Co-Browse
- Web-Based Co-Browse Dynamic Start Page
- Web-Based Co-Browse Initial Start Page
- Web-Based Co-Browse Meet Me
- Web-Based FAQ

Web-Based Voice Callback

This sample demonstrates how to use the callback API (see "Callback" on page 28) to implement voice callback on a web form.

Web-Based Chat

This sample demonstrates how to use the chat API for the Flex Chat protocol (see "Chat" on page 31) to implement a chat feature on a web form.

Web-Based Chat with Statistics

This sample demonstrates how to use the chat API for the Flex Chat protocol (see "Chat" on page 31) and the statistics API (see "Statistics Packages" on page 39) to implement a chat feature and present queue-related statistics on a web form.

Web-Based E-Mail

This sample demonstrates how to use the e-mail API (see "E-Mail" on page 35) to send e-mail using a web form.

Web-Based E-Mail with Attachment

This sample demonstrates how to use the e-mail API (see "E-Mail" on page 35) to send an e-mail with an attachment via a web form.

Web-Based E-Mail with Statistics

This sample demonstrates how to use the e-mail API (see "E-Mail" on page 35) to submit a web form to E-mail Server Java, and to use the statistics API (see "Statistics Packages" on page 39) to retrieve statistics about interactions' queue length, distribution volume, and distribution time from Stat Server.

Web-Based Co-Browse

This sample demonstrates how to use the Co-Browsing Server API to implement co-browse on a web form.

Web-Based Chat and Co-Browse

This sample demonstrates how to use the chat API for the Flex Chat protocol (see "Chat" on page 31) and the Co-Browsing Server API to implement a chat feature with co-browse on a web form.

Web-Based Co-Browse Dynamic Start Page

This sample demonstrates how to use the Co-Browsing Server API to implement a dynamic start page for co-browse on a web form.

Web-Based Co-Browse Init Start Page

This sample demonstrates how to use the Co-Browsing Server API to implement an initial start page for co-browse on a web form.

Web-Based Co-Browse Meet Me

This sample demonstrates how to use the Co-Browsing Server API to implement the meet me feature for co-browse on a web form.

Web-Based FAQ

This sample demonstrates how to implement FAQ functionality on a web form.

Web-Based Open Media

This sample demonstrates how to use the Open Media API (see "Open Media" on page 37) to submit and update an interaction using a web form.

Shared Files

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

- `CommLib.js`—see "Common JavaScript Functions" on page 99.
- `Constants.js`—see "Constants" on page 99.
- `constants.jsp`—see "Constants" on page 99.
- `icc_start.jsp`—see "Load Balancing" on page 100.
- `icc_start_client.jsp`—see "Load Balancing" on page 100.
- `Security.jsp`—see "Masking Text" on page 100.
- `icc_style.css`—see "Style Sheet" on page 101.
- `index.html`—see "Greetings Page" on page 100.
- `arrow.gif`—see "Graphics" on page 101.
- `fon.gif`—see "Graphics" on page 101.
- `genesyslogo-trans.gif`—see "Graphics" on page 101.

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

Two files contain constant values, Constants.js and constants.jsp. The client uses the Constants.js file; the media servers use the constants.jsp file.

The Constants.js file contains values such as the virtual routing point for e-mail and chat requests and the e-mail address for sending form requests.

The constants.jsp file (see Table 8 on page 99) contains values such as:

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

Table 8: Constant Values in constants.jsp

Variable Name	Default Value	Description
fldnPhoneNumber	PhoneNumber	User's phone number
fldnFirstName	FirstName	User's first name
fldnLastName	LastName	User's last name
fldnEmailAddress	EmailAddress	User's e-mail address
fldnFromAddress	FromAddress	E-mail's from address
fldnStartTime	StartTime	Start time
fldnEndTime	EndTime	End time
fldnMedia	Media	The communication medium such as chat, callback, or e-mail

Table 8: Constant Values in constants.jsp (Continued)

Variable Name	Default Value	Description
fldnNow	Now	The current time
fldnRouteInfo	RouteInfo	The routing point information
fldnSubject	Subject	The subject of an e-mail
fldnEmailBody	EmailBody	The body of an e-mail
fldnReplyFrom	Mailbox	The sender of an e-mail
app_root_url	(“/WebAPISamples761”)	The root directory of the Web API Samples
strTenant	Tenant Name Value	The name of the tenant
strEmailQueue	Inbound queue	The e-mail queue
strChatQueue	Chat inbound queue	The chat queue
strQueueKey	“default”	The queue key
strChatStatInterval	“”	Chat statistics interval
strEmailStatInterval	“”	E-mail statistics interval
chatRefreshTimeout	10000	The frequency at which a chat session is refreshed

Load Balancing

The `icc_start.jsp` and `icc_start_client.jsp` files perform load balancing on the server side.

Masking Text

The `Security.jsp` file contains a `mask_html()` method (with different signatures) translates special characters like ", ', <, >, and & into HTML abbreviations like ‘`&lt;`’, ‘`&gt;`’, and ‘`&quot;`’.

Presentation

Greetings Page

The `index.html` 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.html` and various samples. `Arrow.gif` is a graphic of a forward arrowhead. `fon.gif` is a graphic for background wallpaper design. `Genesyslogo-trans.gif` contains the Genesys company logo.

Style Sheet

The cascading style sheet (CSS), `icc_style.css`, has 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.

Callback Sample

This section presents the purpose, functionality overview, and code implementation for the Callback Sample.

Purpose

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

- Connect to Universal Callback Server using the Callback API.
- Submit a callback request.
- Cancel a callback request.
- Get information about a request.
- View a list of the user’s callback requests.

Functionality Overview

The following sections review the code used in implementing the different callback functions:

- “Setting the Content Type and Character Encoding” on page 102
- “Loading Libraries and Importing Files” on page 102
- “Handling Content” on page 102

Files

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

Code Explanation

The following subsections explain the code in `Callback.jsp`. 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 most important points. In these cases, the missing lines have been replaced with "...".

Setting the Content Type and Character Encoding

The first line 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 Universal Callback Server `Application` object in Configuration Server:

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

Loading Libraries and Importing Files

Then the sample code loads the following libraries into memory:

```
<%@ page import="Genesys.webapi.system.loadbalancing.*" %>
<%@ page import="Genesys.webapi.media.callback.direct.*" %>
<%@ page import="Genesys.webapi.media.callback.protocol.*" %>
<%@ page import="Genesys.webapi.media.common.*" %>
<%@ page import="Genesys.webapi.utils.i18n.*" %>
<%@ page import="Genesys.CfgLib.*" %>
```

The code needs to 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

Next the code writes some HTML header tags:

```
<html>
<head>
   <link rel="stylesheet" href="/WebAPISamples761/icc_style.css"
      type="text/css">
   <title>MCR 7.6.1 samples. Callback service</title>
</head>
<body language="JavaScript" onload="return window_onload();
   " background="/WebAPISamples761/fon.gif">
```

```
<h2 align="center">MCR 7.6.1 samples. Callback service.</h2>
<script language="JavaScript">
...
```

Retrieving Parameters

The Java code section retrieves the request parameters, such as the first and last name of a customer, from the `i18nsupport.GetSubmitParametr()` method instead of the `HTTPServletResponse.getParameter()` method. This step is necessary to support foreign languages. For more information, see "International Language Support" on page 46.

```
<%
    String action = i18nsupport.GetSubmitParametr(request, "cmd");
    String first_name = i18nsupport.GetSubmitParametr(request, fldnFirstName);
 ...
```

Getting Load Balancer and Universal Callback Server Instance

This section of code creates a load balancer instance and returns an available instance of Callback Server for each request. It may seem a bit odd to have this step after the parameter retrieval step. This order occurs because the code actually gets new load balancer and Callback Server instances for each request. Hence, for every submission, the JSP gets the new objects.

Due to the simplicity of callback requests, which require only one pass to the Callback Server, your code can disconnect from the load balancing servlet as soon as it receives the new Callback Server instance:

```
// get callback server we are working with
String aliasCallback = i18nsupport.GetSubmitParametr(request, "aliasCallback");

String svcHost = null;
int svcPort = -1;

boolean form_shown = false;

// create load balancer instance
SvcDispatcher svcDispatcher = new SvcDispatcher();
// if callback server is not yet selected...
if( aliasCallback == null || aliasCallback.equals("") )
{
// ... select callback server
   if( svcDispatcher == null || svcDispatcher.getErrorCode() != 0 ||
   !svcDispatcher.inqSrvcByType(CfgAppType.CFGUniversalCallbackServer, strTenant) )
   {
%>
<P align="center">We are sorry, callback service is unavailable at this time.
   Please try again later.</P>
<%
   svcDispatcher = null;
}
else
{
// remember server alias
   aliasCallback = new String(svcDispatcher.getSrvcAlias());
}
}
else
{
// callback server has been already selected, we need to get the info by alias
   if( svcDispatcher == null || svcDispatcher.getErrorCode() != 0 ||
   !svcDispatcher.inqSrvcByAlias(aliasCallback) )
   {
%>
<P align="center">We are sorry, callback service is unavailable at this time.
      Please try again later.</P>
<%
   svcDispatcher = null;
}
}

if( svcDispatcher != null )
{
   svcHost = new String(svcDispatcher.getSrvcHost().toLowerCase());
   svcPort = svcDispatcher.getSrvcPort();

   svcDispatcher = null;   // free dispatcher

   form_shown = true;  // we are drawing the form
%>
```

Constructing the HTML Body

Next, the code includes more HTML code to create input boxes and two types of submit buttons, one for viewing a list of the user's callback requests and the other for sending a callback request:

```
<FORM id=frm_callback name="frm_callback" method="get"
   action="Callback.jsp?aliasCallback=<%=mask_html(aliasCallback)%>">
   <table border="1">
      <tr>
         <td colspan=6>Please enter your personal information:</td>
         ...
         <td colspan="6" align="center">
            <INPUT TYPE="submit" NAME="cmd" VALUE="View list" onclick="return
               on_view_list();">&nbsp;&nbsp;&nbsp;&nbsp;
            <INPUT TYPE="submit" NAME="cmd" VALUE=
               "Request callback" language="JavaScript"
               onclick="return on_request();">&nbsp;&nbsp;&nbsp;&nbsp;
            <INPUT TYPE="reset" VALUE="Clear form">
         </td>
      </tr>
   </table>

   <input type="hidden" id=<%=mask_html(fldnStartTime)
      %> name="<%=mask_html(fldnStartTime)%>">
   <input type="hidden" id=<%=mask_html(fldnEndTime)
      %> name="<%=mask_html(fldnEndTime)%>">
</FORM>
<BR>
```

The Java code next, in the `try-catch` block, attempts to get a handle to the Callback Server. If it is unsuccessful, the `_callback_direct` class returns an error message:

```
<%
if( action != null  && !action.equals("") )
{
   _callback_direct callback = null;

   try
   {
      callback = new _callback_direct(svcHost, svcPort);
   }
   catch(_communication_exception ex)
   {
%>
<p align="center">We are sorry, callback service is unavailable at this time.
      Please try again later. </P>
<%
   out.write(ex.toString());
   action = "";    // no more actions
}
```

Tracking States and Submission

The JSP checks its own state after a submission, because the JSP calls itself in response to the user events. This occurs in several places in `Callback.jsp`. By way of example, here is what happens when a user requests a callback:

Note: The sample JSP keeps track of its own *state* based on user interaction.

```
if( action.equals("Request callback") )
{
   _kvlist userdata = new _kvlist();
   userdata.addElement(new _kvitem(fldnFirstName, first_name));
   userdata.addElement(new _kvitem(fldnLastName, last_name));
   userdata.addElement(new _kvitem(fldnEmailAddress, email_address));
   userdata.addElement(new _kvitem(fldnRouteInfo, "default"));
   if (media.equals("voice"))
   userdata.addElement(new _kvitem("MediaType", "WebCall"));// just an example
   else
```

```
  userdata.addElement(new _kvitem("MediaType", "WebCallIP"));// just an example
  int rc;
  if( now.equals("Immediate") )
  {
    // immediate callback, no times required
    rc = callback.request(phone_number, media, userdata);
  }
  else
  {
    // scheduled callback
    rc = callback.request(phone_number, media, request.getParameter(fldnStartTime),
      request.getParameter(fldnEndTime), userdata);
  }

  if( _callback_direct.__rc_ok == rc )
  {
%>
  <p>Callback request has been successfully submitted.<br>
  Request identifier: <%=mask_html(callback.reqid())%></p>
<%
}
else
{
%>
  <p>Failed to submit callback request. Reason:
   <%=mask_html(callback.lasterror())%></p>
<%
}

// show list of callback requests with new request
action = "View List";
}
```

Handling User Events

The following JavaScript functions handle user events:

- `ConvertTime(UTCtime)`—Converts the specified UTC time to the equivalent local date and time.
- `window_onload()`—Called every time the page is loaded. This function currently sets the form's default callback time to the time the page was loaded, but you may also modify it to execute any other tasks that should be carried out whenever the page is loaded.
- `on_view_list()`—Triggered by the `View List` Submit action. Returns `false` if the user has not entered a contact number, thus preventing the submission of the form.
- `on_request()`—Triggered by the `Request Callback` Submit action. Returns `false` if the user has not entered a contact number, thus preventing the submission of the form. Otherwise, it generates a start and end date for the callback request.

- `GetUTCTime(date)`—Converts the specified `date` to the equivalent UTC time.
- `double_digit(Value)`—Ensures that the `Value` passed to the function contains two digits.
- `setSelection(objControl, Value)`—Sets the selected value of a document field (`ObjControl`) to `Value`.
- `getSelectedOption(opt)`—Takes a collection of options for a field and returns the option that has been selected by the user.
- `on_reset()`—Resets the form to its default values.

Chat Sample

The .../`WebAPISamples761` directory contains files for the Chat (also known as web-based chat), the Chat with Statistics, and the Chat and Co-Browse Samples.

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

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 used in implementing the different chat functions:

- "Setting the Content Type and Character Encoding" on page 111
- "Loading the Required Libraries and Files" on page 111
- "Connecting to Chat Server" on page 111
- "Creating a Chat Session" on page 112
- "Handling Content" on page 113
- "Tracking State" on page 116
- "Closing the Connection" on page 117

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.

- `HtmlChatFrameSet.jsp`—A frameset that holds the `HtmlChatCommand.jsp` and `HtmlChatPanel.jsp` files.
- `HtmlChatPanel.jsp`—A file that contains 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 code resides in the `HtmlChatPanel.jsp` file. The code is divided into these functions:

- "Setting the Content Type and Character Encoding"
- "Loading the Required Libraries and Files"
- "Handling Content"

Setting the Content Type and Character Encoding

First, the sample JSP sets the content type and character encoding:

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

Loading the Required Libraries and Files

The JSP then loads the following library into memory:

```
<%@ page import="Genesys.webapi.utils.i18n.*" %>
```

The code needs access to other common features like constants and security checks, which are contained in these files:

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

Handling Content

This section covers these topics:

- "Writing HTML Headers"
- "Importing JavaScript Functions and Constants"
- "Handling User Events"

Writing HTML Headers

```
<html>
<head>
<link rel="stylesheet" href="/WebAPISamples761/icc_style.css"
type="text/css">
<title>Compound sample 7.6.1 Chat service</title>
</head>
```

Importing JavaScript Functions and Constants

The code uses the common JavaScript functions and constants:

```
<SCRIPT LANGUAGE=javascript
SRC="/WebAPISamples761/CommLib.js"></SCRIPT>
<SCRIPT LANGUAGE=javascript
SRC="/WebAPISamples761/Constants.jsp"></SCRIPT>
```

Handling User Events

The `HtmlChatPanel.jsp` file's body begins with a call to this function:

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

The `HtmlChatCommand.jsp` file 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.
- `on_refresh()`—Calls the `on_refresh()` function on the page residing in the `itf` frame.
- `CommandFrameReady()`—Boolean flag for making the frame ready to accept a command.
- `AddMessage()`—Appends new text to the chat transcript.
- `show_message()`—Displays the latest chat transcript.
- `SetSessionID()`—Returns the session ID.
- `message_onkeypress()`—The event processor for "user is typing" notification. It sends a notification when the user starts typing into "message" field.

```
<INPUT onkeypress="javascript:message_onkeypress();" TYPE="String"
size="60" id="message" NAME="message"></td>
```

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

- `window_onunload()`
- `clear_transcript()`
- `doNothing()`
- `disconnected()`
- `connected()`

Finally, the HTML code creates input boxes for users to fill in their personal information and their message. The code also creates submit buttons so users can send their new chat messages:

```
<FORM name="chat_form" id=chat_form onSubmit="javascript:on_send();
     return false;">
<table border="1">
<tr>
    <td colspan=6>Personal information:</td>
</tr>
<tr>
    <td>First name:</td>
    <td><INPUT TYPE="String"
NAME="<%=mask_html(fldnFirstName)%>"></td>
    <td>Last name:</td>
    <td><INPUT TYPE="String"
NAME="<%=mask_html(fldnLastName)%>"></td>
    <td>E-mail address:</td>
    <td><INPUT TYPE="String"
NAME="<%=mask_html(fldnEmailAddress)%>"></td>
</tr>
<tr>
    <td colspan=6 align="center">
     <A href="javascript:on_connect();" >Start chat</a>
       &nbsp;&nbsp;&nbsp;
       <A href="javascript:on_disconnect();" >Stop chat</a>
    </td>
</tr>
<tr>
    <td colspan=6>
        <textarea cols="80" rows="10" id="transcript"
NAME="transcript"></textarea>
    </td>
</tr>
<tr>
    <td>Message:</td>
    <td colspan="2">
     <input onkeypress= "javascript:message_onkeypress();"
       type="text"name="message" size="60"/>
   </td>
   <td>
     <input type="submit" id="send" value="Send" name="send"/>
   </td>
</tr>...
</body>
</html>
```

Logic Implementation

The HtmlChatCommand.jsp contains the main logic for the Chat Sample's functionality. The subsections below explain the code in the file.

Setting the Content Type and Character Encoding

The first line in the sample 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 Chat Server Application object in Configuration Server:

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

Loading the Required Libraries and Files

The sample code then loads the following libraries into memory:

```
<%@ page import="Genesys.webapi.system.loadbalancing.*" %>
<%@ page import="Genesys.webapi.media.chat.direct.*" %>
<%@ page import="Genesys.webapi.media.chat.protocol.*" %>
<%@ page import="Genesys.webapi.media.common.*" %>
<%@ page import="Genesys.webapi.utils.i18n.*" %>
<%@ page import="Genesys.CfgLib.*" %>
```

The code also needs access to other JSP files. Here, it includes the constants.jsp and Security.jsp files:

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

Connecting to Chat Server

Chat Server works differently from E-mail Server Java, whose behavior is described in “E-Mail Sample” on page 120. The key difference is that Chat Server must maintain a live connection between users. (In e-mail, users fill out a form. Once they submit the form, the transaction is over. Chat Server, however, must maintain each user's identity until one party ends the session.)

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

```
if( cmd.equals("connect")){
   if( first_name.equals("") || last_name.equals("")){
      itf_response    =  "USERNAMEREQUIRED";
      itf_message     =   "Please enter first and last names.";
```

Likewise, if the load-balancing servlet cannot return an available Chat Server, the sample code informs the user:

```
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.";
```

If the load balancer finds a Chat Server, the code returns an alias to that Chat Server, then tries to connect and log the user in:

```
   // we have a chat server at this point
   chat_alias  =   svcDispatcher.getSrvcAlias();
   try{
      _chat_direct chat =
                   new _chat_direct(svcDispatcher.getSrvcHost(),
                                      svcDispatcher.getSrvcPort());
      _kvlist userdata = new _kvlist();
      userdata.addElement(new _kvitem(fldnFirstName, first_name));
      userdata.addElement(new _kvitem(fldnLastName, last_name));
      userdata.addElement(new _kvitem(fldnEmailAddress,
email_address));

      userdata.addElement(new _kvitem(fldnVRP, vrp));
      String strNickName = "";
      if (last_name != null && last_name.length() > 0)
        strNickName = first_name + last_name.charAt(0);
      else
        strNickName = first_name;
      int rc = chat.login(strNickName, userdata, timeZoneOffset);
```

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

```
   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!";
```

Creating a Chat Session

Next, the code calls the join() method in an attempt to create a chat session. If Chat Server returns an ok status, the user joins the new session successfully. Otherwise, the JSP returns an error description received from Chat Server:

```
rc = chat.join(user_id, secure_key, "", "Some Subject");
if( _chat_direct.__rc_ok == rc && chat.user_id() != null &&
     chat.secure_key() != null && chat.transcript() != null){
   clear_transcript    =   true;
   transcript = chat.transcript();
   script_pos = chat.script_pos();
   itf_response = "CONNECTED";
}else{
   itf_response = "DISCONNECTED";
   if( chat.errdesc() != null )
      itf_message = chat.errdesc();
   else
      itf_message = "Could not create a chat session.";
}
```

Handling Content

This section covers these topics:

- "Loading Parameters"
- "Handling User Requests"
- "Masking Data"
- "Processing Server Events"
- "Handling User Events"

Loading Parameters

The `HtmlChatCommand.jsp` code loads the hidden parameters on 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 topic "Tracking State" on page 116 for more information:

```
<script language="JavaScript">
<%
    // 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 = "";

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

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

String script_pos = i18nsupport.GetSubmitParametr(request, "script_pos");
if( script_pos == null ) script_pos = "1";

String msg2send = i18nsupport.GetSubmitParametr(request, "msg2send");
if( msg2send == null ) msg2send = "";
```

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;

if (svcDispatcher == null)
   svcDispatcher = new SvcDispatcher();
```

Handling User Requests

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

```
} else if( !chat_alias.equals("")){
   try {
      if( svcDispatcher.inqSrvcByAlias(chat_alias)){
         _chat_direct chat =
                new _chat_direct(svcDispatcher.getSrvcHost(),
                                svcDispatcher.getSrvcPort());

         if( cmd.equals("disconnect") ){
            chat.logout(user_id, secure_key);
            itf_response    =   "DISCONNECTED";
            itf_message     =   "Chat was finished";
         }
         if( cmd.equals("send") ){
            ...
         }
```

The send and user_typing commands are similar except for the last parameter passed by user_typing. The value of the last parameter should be _chat_packet.__attrv_treat_as_system, this will ensure that the message is not displayed to the client's side.

Masking Data

The MaskSymbols() method filters out any characters deemed unacceptable by the servers:

```
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')
         ...
      }
   return strOut;
}
```

Processing Server Events

When the sample receives a server event, the code calls the event_type() 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 (event.event_type().equals(
                      _chat_packet.__attrv_event_type_connect)){

   text2append = text2append + "New party ('" + event.user_nick() +
                                      "') has joined the session";

}else if (event.event_type().equals(
                      _chat_packet.__attrv_event_type_message)){
      if (event.event_body() != null)
         text2append = text2append + event.user_nick() + ": " +
                      event.event_body();
      else
         text2append = text2append + event.event_body() + ":";
   } else if (event.event_type().equals(
                      _chat_packet.__attrv_event_type_abandon)){

         text2append = text2append + "Party ('" + event.user_nick() +
                      "') has left the session. Reason: " +
                      event.event_body();
      }
```

Handling User Events

Five JavaScript functions 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()—displays a system message on the agent's desktop, but does not display on the user's screen.

Tracking State

The 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="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)%>">
```

Some of these hidden variables are straightforward. 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 must reset the service dispatcher to `null`. Otherwise, there will be orphan connections to the server that do not relinquish their resources (principally network resources, like sockets, and memory):

```
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 Statistics Sample

This section presents the purpose, functionality overview, files, and code explanation for the Chat with Statistics Sample.

Note: The difference between this sample and the Chat Sample is the additional *statistics* feature. Therefore, this section discusses only the statistics portions of the sample code. For shared code that provides the chat features for both samples, see "Code Explanation" on page 108.

Purpose

The sample demonstrates how to add the following to a web form that supports Java libraries:

- A simple chat feature
- A statistics feature that displays data such as queue position and estimated wait time

Functionality Overview

The following sections review the implementation of the statistics functions:

- "Setting the Content Type and Character Encoding" on page 118
- "Loading the Required Libraries and Files" on page 118
- "Connecting to Stat Server" on page 119
- "Getting the Statistics" on page 119

Files

The .../ChatWithStatistic directory represents the HTML Chat with Statistics Sample. The sample consists of five files:

- blank.jsp—A file used as a placeholder
- ChatStatInfo.jsp—A file that requests statistics using the statistics API
- 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 to create a chat panel

Code Explanation

The ChatStatInfo.jsp file contains the main logic for the sample's statistics functionality. The rest of this section explains only the code from that file. Note that blank.jsp is only a placeholder. The code is explained in these subsections:

- "Setting the Content Type and Character Encoding"
- "Loading the Required Libraries and Files"
- "Connecting to Stat Server"
- "Getting the Statistics"

Setting the Content Type and Character Encoding

The first line in the sample 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 Chat Server Application object in Configuration Server:

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

Loading the Required Libraries and Files

The sample then loads the following libraries into memory:

```
<%@ page import="Genesys.webapi.system.loadbalancing.*"%>
<%@ page import="Genesys.webapi.media.common.*"%>
<%@ page import="Genesys.webapi.stat.direct.*"%>
<%@ page import="Genesys.webapi.stat.protocol.*"%>
<%@ page import="Genesys.webapi.utils.i18n.*"%>
<%@ page import="Genesys.CfgLib.*"%>
```

The code needs access to other JSP files. Here it includes `constants.jsp`:

```
<%@ include file="../constants.jsp" %>
```

Connecting to Stat Server

The code loads and retrieves these parameters from the `HTTPServletRequest` on each JSP refresh:

```
String strAction = request.getParameter("action");
if( strAction == null ) strAction = "";

String strJsWindow = request.getParameter("jswindow");
if( strJsWindow == null || 0 == strJsWindow.length()) strJsWindow = "parent.";
```

Next, the code checks its instance of `SvcDispatcher`—the load-balancing servlet. If the load balancer finds a Stat Server and is able to connect to it, then the JSP can proceed:

```
if (svcDispatcher != null && svcDispatcher.getErrorCode() == 0 &&
   svcDispatcher.inqSrvcByType(CfgAppType.CFGStatServer, strTenant))
{
  svcHost = new String(svcDispatcher.getSrvcHost().toLowerCase());
  svcPort = svcDispatcher.getSrvcPort();
```

Getting the Statistics

If the requested action is `getQueueStat`, the JSP tries to retrieve the chat queue length, the total distribution time, and the total number of distributed chat sessions. The JSP then uses the total distribution time and the total distributed sessions to calculate the estimated wait time:

```
if (strAction.equals ("getQueueStat"))
{
  try
  {
    iQueueLength = Integer.parseInt(SD.getQueueStat(svcHost, svcPort,
      strTenant, strChatQueue, stat_direct._stat_chat_queue_length,
      strChatStatInterval));
    iTotalDistributionTime = Integer.parseInt(SD.getQueueStat(svcHost,
      svcPort, strTenant, strChatQueue,
      stat_direct._stat_chat_total_distribution_time,
      strChatStatInterval));
    iTotalDestributed = Integer.parseInt(SD.getQueueStat
      (svcHost, svcPort, strTenant, strChatQueue,
      stat_direct._stat_chat_total_destributed, strChatStatInterval));
    if (iTotalDestributed != 0)
      iEWT = iTotalDistributionTime/iTotalDestributed;
  }
```

If an exception occurs, it is printed to a log file, and the JSP invokes some error-handling routines:

```
      catch (Exception e)
      {
         System.out.println (e.toString());
            bError = true;
            strError = e.toString();
%>
      <script>
         <%=strJsWindow%>OnError("<%=strError%>");
      </script>
<%
      }
   }
   if (bError == false)
   {
%>
   <script>
      <%=strJsWindow%>OnQueueStatInfo("<%=iQueueLength%>",
      "<%=ConvertSecondsToString(iEWT)%>");
   </script>
<%
   }
}
%>
```

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 thing that sets this sample apart from an ordinary or external e-mail is that the web form e-mail goes through E-mail Server Java, which routes the e-mail to the appropriate agent using Genesys Universal Routing Server.

The sample is a JSP file that shows how to:

- Connect to E-mail Server Java using the E-mail API.
- Submit an e-mail request to E-mail Server Java.
- Report any possible errors in the sample.
- Disconnect from E-mail Server Java.
- Gather submitted information.

- Communicate with the load-balancing service.
- Communicate with E-mail Server Java.
- Generate HTML responses and JavaScript code based on responses from E-mail Server Java.
- Mask all potentially dangerous server-side data or symbols such as `<`, `>`, `"`, and `'`.

Functionality Overview

The following sections review the code used in implementing the different e-mail functions:

- "Setting the Content Type and Character Encoding" on page 121
- "Loading Libraries and Importing Files" on page 121
- "Handling Content" on page 122
- "Closing the Connection" on page 124

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 `Email.jsp`.

Setting the Content Type and Character Encoding

The first line 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 E-mail Server Java `Application` object in Configuration Server:

```
<%response.setContentType(Ch. 9: UNIX; chg. "resource" warnings on pgs. 110, 117, 123
   to match 170. charset=" + i18nsupport.GetCharSet());%>
```

Loading Libraries and Importing Files

Then, the sample code loads the following libraries into memory:

```
<%@ page import="Genesys.webapi.system.loadbalancing.*"%>
<%@ page import="Genesys.webapi.media.irs.direct.*"%>
<%@ page import="Genesys.webapi.media.irs.protocol.*"%>
<%@ page import="Genesys.webapi.media.common.*"%>
<%@ page import="Genesys.webapi.utils.i18n.*"%>
<%@ page import="Genesys.CfgLib.*"%>
<%@ page import="java.net.URLEncoder.*"%>
```

The code needs to 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

Next the code writes some HTML header tags:

```
<html>
<head>
<link rel="stylesheet" href="/WebAPISamples761/icc_style.css" type="text/css">
<title>MCR 7.6.1 samples. E-mail over the Web sample</title>
</head>
<body LANGUAGE="javascript" onload="window_onload();" onunload="window_onunload();"
   background="/WebAPISamples761/fon.gif">
<SCRIPT LANGUAGE=javascript SRC="/WebAPISamples761/CommLib.js"></SCRIPT>
<H2>MCR 7.6.1 samples. E-mail over the Web sample</H2>
```

Retrieving Parameters

This Java code section retrieves the request parameters, such as the first and last name of a customer, from the i18nsupport.GetSubmitParametr() method instead of the HTTPServletResponse.getParameter() method. This step is necessary to support foreign languages. For more information on language support, see “International Language Support” on page 46:

```
<%
    String action = i18nsupport.GetSubmitParametr(request, "action");
    String first_name = i18nsupport.GetSubmitParametr(request, fldnFirstName);
 ...
```

Getting Load Balancer and E-Mail Server Java Instance

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

```
SvcDispatcher svcDispatcher = new SvcDispatcher();
if( svcDispatcher != null && svcDispatcher.getErrorCode() == 0
        && svcDispatcher.inqSrvcByType(
             CfgAppType.CFGEmailServer, strTenant)){

   String svcHost = new
              String(svcDispatcher.getSrvcHost().toLowerCase());
   int svcPort = svcDispatcher.getSrvcPort();
%>
```

Constructing the HTML Body

Next, the code includes more HTML code to create input boxes:

```
<FORM id=frm_irs name="frm_irs" method="post" action="Email.jsp"
   onSubmit="JavaScript:return Submit_onClick (1);">
<table border="1">
   <tr>
    <td colspan=6>Please enter the interaction information:</td>
   ...
   </tr>
</table>
</FORM>
<BR>
```

The Java code's `try-catch` block now attempts to get a handle to E-mail Server Java. If it does not succeed, the `_irs_direct` class returns an error message:

```
<%
if( action != null && !action.equals("") ){
   _irs_direct irs = null;
   try {
      irs = new _irs_direct(svcHost, svcPort);
   }catch(_communication_exception cex) {
%>
      We are sorry, E-mail Server is temporary down,
            please try again later.<br>
<%
      out.write(cex.toString());
      action = "";    // no more actions
   }
```

Tracking States and Submission

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. Unlike in HTML, the JSP file must be submitted back to the server to get changes even if the user is not really ready to submit the form. Hence, the JSP submit is not the same as an HTML submit. The Sun Microsystems website has tutorials that explain how JSPs work:

Note: The sample JSP keeps track of the *state* of the file based on user interaction.

```
   if( action.equals("Submit") ){
      // submit the new interaction to E-mail server
      ...
   }
```

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 form ID value to the request ID value.
- `window_onunload()`—has no code. It is an empty function.

- `Submit_onClick()`—sets the action flag 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. 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.

Closing the Connection

When the user is ready to submit the form, the connection to E-mail Server Java must be closed 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.

```
   if( irs != null ){
      irs.close();
      irs = null;
   }
}...
svcDispatcher = null;   // free dispatcher
%>
```

E-Mail with Attachment Sample

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

Purpose

The E-mail with Attachment Sample code demonstrates how a user can send an e-mail with an attachment via a web form.

Files

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

Code Explanation

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

Setting the Content Type and Character Encoding

The first line in the `Email.jsp` file sets the page content type:

```
<%@ page contentType = "text/html; charset="windows-1252" %>
```

The second line is a call to the `response.setContentType()` method. This sets the content type or character encoding to use in the response to the client:

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

Loading Libraries and Importing Files

Then, the sample code loads the following libraries into memory:

```
<%@ page import="java.io.*"%>
<%@ page import="java.util.*"%>
<%@ page import="org.apache.commons.fileupload.*"%>
<%@ page import="org.apache.commons.fileupload.disk.*"%>
<%@ page import="org.apache.commons.fileupload.portlet.*"%>
<%@ page import="org.apache.commons.fileupload.servlet.*"%>
<%@ page import="org.apache.commons.io.*"%>
<%@ page import="Genesys.webapi.system.loadbalancing.*"%>
<%@ page import="Genesys.webapi.media.irs.direct.*"%>
<%@ page import="Genesys.webapi.media.irs.protocol.*"%>
<%@ page import="Genesys.webapi.media.common.*"%>
<%@ page import="Genesys.webapi.utils.i18n.*"%>
<%@ page import="Genesys.CfgLib.*"%>
<%@ page import="java.net.URLEncoder.*"%>
```

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

The code needs to 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

Next, the code writes some HTML header tags:

```
<html>
<head>
<link rel="stylesheet" href="/WebAPISamples761/icc_style.css"
type="text/css">
<title>MCR 7.6.1 samples. E-mail over the Web sample</title>
</head>

<body LANGUAGE="javascript" onload="window_onload();"
onunload="window_onunload();"
background="/WebAPISamples761/fon.gif">
<SCRIPT LANGUAGE=javascript
SRC="/WebAPISamples761/CommLib.js"></SCRIPT>
<H2>MCR 7.6.1 samples. E-mail over the Web sample</H2>
```

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   = "";
```

Getting Load Balancer and E-Mail Server Java Instance

This next section of code creates a load balancer instance and returns an available instance of E-mail Server Java for each request:

```
// create load balancer instance
SvcDispatcher svcDispatcher = new SvcDispatcher();

// We select IRS server for each request(!)
if( svcDispatcher != null && svcDispatcher.getErrorCode() == 0 &&
   svcDispatcher.inqSrvcByType(CfgAppType.CFGEmailServer, strTenant)
)
    {
        String svcHost = new
String(svcDispatcher.getSrvcHost().toLowerCase());
        int svcPort = svcDispatcher.getSrvcPort();
```

Constructing HTML Body

Next, the code includes more HTML code to create input boxes and buttons. The first line of code sets the form's `enctype` attribute 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. There are two fields at the bottom of the form where file names can be entered and attached to the e-mail:

```
<FORM enctype="multipart/form-data" id=frm_irs name="frm_irs" method="post"
   action="Email.jsp" onSubmit="JavaScript:return Submit_onClick (1);">
<table border="1">
<tr>
    <td colspan=6>Please enter the interaction information:</td>
</tr>
<tr>
   <td>First name:</td>
   <td><INPUT TYPE="String" NAME="<%=mask_html(fldnFirstName)%>"
     VALUE="<%=mask_html(first_name)%>">
   </td>
   <td>Last name:</td>
   <td><INPUT TYPE="String" NAME="<%=mask_html(fldnLastName)%>"
     VALUE="<%=mask_html(last_name)%>">
   </td>
   <td>E-mail address:</td>
   <td><INPUT TYPE="String" NAME="<%=mask_html(fldnFromAddress)%>"
     VALUE="<%=mask_html(email_address)%>">
   </td>
</tr>
<tr>
   <td>Reply from:</td>
   <td colspan=2><INPUT TYPE="String" NAME="<%=mask_html(fldnReplyFrom)%>"
     VALUE="<%=mask_html(reply_from)%>">
   </td>
   <td>Interaction id:</td>
   <td colspan=2><INPUT SIZE="40" TYPE="String" NAME="id"
     VALUE="<%=mask_html(id)%>">
   </td>
</tr>
<tr>
   <td>Subject:</td>
   <td colspan=5><INPUT SIZE="80" TYPE="String"
     NAME="<%=mask_html(fldnSubject)%>"
     VALUE="<%=mask_html(subject)%>">
   </td>
</tr>
```

```
<tr>
   <td colspan=6>
     <textarea cols="100" rows="10"
        NAME="<%=mask_html(fldnEmailBody)%>">
        <%=mask_html_for_textarea(body)%>
     </textarea>
   </td>
</tr>
<tr>
    <td colspan=2> File to upload: </td>
    <td colspan=4>
     <input type="file" name="attachment1">
    </td>
</tr>
<tr>
    <td colspan=2> File to upload: </td>
    <td colspan=4>
     <input type="file" name="attachment2">
    </td>
</tr>
<tr>
    <td colspan="6" align="center">
     <input type="hidden" name="action" value="">

     <A HREF="JavaScript:Submit_onClick(0);">Submit</A>
     <A HREF="JavaScript:Reset_onClick();">Reset</A>
    </td>
</tr>
</table>
</FORM>
```

Upload the Multipart File

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

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

          // check if current item is form field or uploaded file
          if(item.isFormField())
          {
            // get the name of the field
           String fieldName = item.getFieldName();
           String fieldValue =
              item.getString(i18nsupport.GetCodePage());
// if a name, we can set it in request to thank the user

            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();
```

Below, the `FilenameUtils.getName()` function is used to return the full filename. Only the text after the last forward slash or backslash will be returned. This function handles both UNIX and Windows filename 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();
             }
          }
        }
      }
      catch (Exception e)
      {
        strFileUploaderError = "Can't process Web form submission.
Exception occured: <BR>" + e.toString();
      }...
```

Tracking States and Submission

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 found in the E-mail Sample, except that additional code has been added to include the attachments in the submission:

```
if( action.equals("Submit") )
  {
    // submit the new interaction to E-mail server
    _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 (attachment1 != null || attachment2 != null)
    {
      _kvlist kvAttachments = new _kvlist();

      if(attachment1 != null && attachment1.length>0)
      {
        _kvlist kvAttachment1 = new _kvlist();
        kvAttachment1.addElement(new _kvitem("Content", attachment1));
        kvAttachment1.addElement(new _kvitem("ContentType",contenttype1));
        kvAttachments.addElement(new _kvitem(filename1, kvAttachment1));
      }
      if(attachment2 != null && attachment2.length>0)
      {
        _kvlist kvAttachment2 = new _kvlist();
        kvAttachment2.addElement(new _kvitem("Content", attachment2));
        kvAttachment2.addElement(new _kvitem("ContentType",contenttype2));
        kvAttachments.addElement(new _kvitem(filename2, kvAttachment2));
      }
      userdata.addElement(new _kvitem("Attachments", kvAttachments));
    }

    if (reply_from != null && reply_from.equals("") == false)
      userdata.addElement(new _kvitem(fldnReplyFrom, reply_from));

    if(irs.submit(_media.__media_email, userdata) == irs.__rc_ok)
      {
        id = new String(irs.reqid());...
```

An appropriate message will be displayed based on whether the submission was successful or not:

```
   Request (<%=mask_html(id)%>) has been successfully submitted to
E-mail
            Serve.
</p>
   <%
      }else{
   %>
<p>
   Could not submit the e-mail to E-mail Server.<br>
   Reason: <%=mask_html(irs.lasterror())%>
</p>
   <%
      action = "";    // no more actions
   }
}...
```

Closing the Connection

When the user is ready to submit the form, the connection to E-mail Server Java must be closed 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.

```
   ...if( irs != null )
      {
         irs.close();
         irs = null; // kill the IRS "representative" if any
      }
   }
}else{
      %>
         <P>
            We are sorry, E-mail Server is unavailable at this time.
Please try
               again later.
         </P>
      <%
      }
svcDispatcher   =   null;   // free dispatcher
```

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 form ID value to the request ID value.
- `window_onunload()`—Has no code. It is an empty function.

- `Submit_onClick()`—Sets the action flag to `Submit`. Because this is a real submission from the user, 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 opens 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 that was entered.

E-Mail with Statistics Sample

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

Note: The difference between this sample and "E-Mail Sample" on page 120 is the additional *statistics* feature. Therefore, this section discusses only the statistics portions of the sample code. (These code portions are similar to those in "Chat with Statistics Sample" on page 117.) For shared code that provides e-mail features, see "Code Explanation" on page 121.

Purpose

The E-mail with Statistics Sample code demonstrates an application that submits its user's web-form input to E-mail Server Java, and also retrieves statistics about the number of interactions in a queue (queue length), distribution volume, and distribution time from Stat Server.

Files

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

Code Explanation

This section traces the statistics portions of the the `Email.jsp` file code in the following subsections:

- "Loading Required Libraries"
- "Declaring and Initializing Variables"
- "The GetQueueStatistic_onClick() Function"
- "Connecting to Stat Server"
- "Getting the Statistics"

- “Supporting Functions”

Loading Required Libraries

To support its statistics features, this sample’s `Email.jsp` file loads two libraries that the basic E-mail Sample’s `Email.jsp` file does not load:

```
<%@ page import="Genesys.webapi.stat.direct.*"%>
<%@ page import="Genesys.webapi.stat.protocol.*"%>
```

Declaring and Initializing Variables

This sample declares the following block of variables to receive or support retrieved statistics:

```
String strError = "";
bError = false;
int iQueueLength = 0;           //double2int
int iTotalDistributionTime = 0; //minutes2int
int iTotalDestributed = 0;      //double2int
int iEWT              = 0;
stat_direct SD = new stat_direct ();
String svcHost = "";
int svcPort = -1;
```

The GetQueueStatistic_onClick() Function

At the end of the `FORM` definition, the following statement displays a link to the `GetQueueStatistic_onClick()` function:

```
<A HREF="JavaScript:GetQueueStatistic_onClick();">
   Get E-mail queue statistic</A>
```

This function is defined later in the file, shortly above the page’s closing `</script>` and `</body>` tags. It defines and submits the following action:

```
function GetQueueStatistic_onClick()
{
   document.forms[0].action.value = "get queue statistic";
   document.forms[0].submit();
}
```

Connecting to Stat Server

The code executed in response to that action appears earlier in the file. It checks the `SvcDispatcher` (load-balancing servlet) instance. If the load balancer finds a Stat Server and is able to connect to it, then the JSP can proceed:

```
if(action.equals("get queue statistic"))
{
   if(svcDispatcher != null && svcDispatcher.getErrorCode() == 0 &&
      svcDispatcher.inqSrvcByType(CfgAppType.CFGStatServer,
strTenant))

   {
      svcHost = new
String(svcDispatcher.getSrvcHost().toLowerCase());
      svcPort = svcDispatcher.getSrvcPort();
```

Getting the Statistics

The next code block attempts to retrieve the e-mail queue length, the total distribution time, and the total number of distributed interactions. It then divides the total distribution time by the total number of distributed interactions to calculate the estimated wait time, `iEWT`:

```
try
{
   SD.setLogServer(svcDispatcher.getLogServer());
   iQueueLength = Integer.parseInt(SD.getQueueStat(svcHost, svcPort,
      strTenant, strEmailQueue,
stat_direct._stat_webform_queue_length,
      strEmailStatInterval));
   iTotalDistributionTime =
Integer.parseInt(SD.getQueueStat(svcHost, svcPort,
      strTenant, strEmailQueue,
      stat_direct._stat_webform_total_distribution_time,
      strEmailStatInterval));
   iTotalDestributed = Integer.parseInt(SD.getQueueStat(svcHost,
svcPort, strTenant, strEmailQueue,
stat_direct._stat_webform_total_destributed,
strEmailStatInterval));
   if (iTotalDestributed != 0)
      iEWT = iTotalDistributionTime/iTotalDestributed;
}
```

If an exception occurs, the JSP prints the error to a log file and also displays it on the page:

```
   catch (Exception e)
   {
      System.out.println (e.toString());
      bError = true;
      strError = e.toString();
```

```
%>
<p>
Could not get queue statistic from E-mail Server.<br>
Reason: <%=strError%>
</p>
<%
   }
```

Otherwise, the following code builds a table and displays the retrieved statistics on the page:

```
if (bError == false)
{
%>
<table border=1>
  <tr>
    <td colspan=2>
    Queue statistic
    </td>
  </tr>
  <tr>
    <td>
    Queue size
    </td>
    <td>
    <%=iQueueLength%>
    </td>
  </tr>
  <tr>
    <td>
    Estimated waiting time
    </td>
    <td>
    <%=ConvertSecondsToString(iEWT)%>
    </td>
  </tr>
</table>
```

Supporting Functions

The private `ConvertSecondsToString()` function, defined at the bottom of the file, is used in the calculation of the estimated wait time. Just above the `ConvertSecondsToString()` function is its own supporting function, `PositiveSub()`:

```
private int PositiveSub (int i1, int i2)
{
   int i3 = i1-i2;
   if (i3 > 0 )
```

```
     return i3;
   return 0;

}

private String ConvertSecondsToString(int iSeconds)
{
        int iDays=iSeconds/(60*60*24);
        iSeconds=PositiveSub (iSeconds, iDays*(60*60*24));

        int iHours=iSeconds/(60*60);
        iSeconds=PositiveSub(iSeconds, iHours*(60*60));

        int iMinutes=iSeconds/60;
        iSeconds=PositiveSub(iSeconds, iMinutes*60);

        String strOut = "";

        if (iDays != 0)
   {
     strOut = strOut + iDays + " day";
     if (iDays > 1)
        strOut = strOut + "s";
     strOut = strOut + " ";
   }

        if (iHours != 0)
   {
     strOut = strOut + iHours + " hour";
     if (iHours > 1)
        strOut = strOut + "s";
     strOut = strOut + " ";
   }
        if (iMinutes != 0)
   {
     strOut = strOut + iMinutes + " minute";
     if (iMinutes > 1)
        strOut = strOut + "s";
     strOut = strOut + " ";
   }
        if (iSeconds != 0)
   {
     strOut = strOut + iSeconds + " second";
     if (iSeconds > 1)
        strOut = strOut + "s";
     strOut = strOut + " ";
   }
   if (strOut.equals(""))
     strOut = "not calculated yet";
        return strOut;
}
```

Co-Browse Samples Overview

This section outlines files common to all the Co-Browse Samples.

Common Files

The Co-Browse, the Co-Browse with Initial Start Page, the Co-Browse with Meet Me, and the Chat and Co-Browse samples all use certain common files:

- `hbmessaging.js`—JavaScript file containing the API for Co-Browsing Server.
- `qstring.js`—JavaScript file containing a utility class.
- `hbmessage_to_var.js`—Messaging file that provides messages from the co-browse frame to the web application frame.
- `blank.html`—Default blank page for initializing empty frames used by the API.
- `hbapi.html`—Supports the client-side API and the co-browse 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 co-browse 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 listed in "Related Resources" on page 17. Genesys licenses certain Response Live (formerly Hipbone) co-browsing components from KANA Software, Inc.

Co-Browse Sample

This section outlines the purpose, functionality, and code implementation of the basic Co-Browse Sample.

Purpose

The Co-Browse Sample code demonstrates basic co-browse functionality using the Co-Browsing Server API. This sample includes two files beyond those covered in "Common Files" above: `CoBrowse.htm`, which sets up the display frame; and `CoBrowseEventHandler.jsp`, which contains the sample's logic.

The CoBrowseEventHandler.jsp File Explained

Getting Load Balancer and Co-Browse Server Instances

The `CoBrowseEventHandler.jsp` file's initial `try-catch` script block creates a load-balancer instance and attempts to discover a Co-Browse Server host:

```
try{
   SvcDispatcher svcDispatcher = new SvcDispatcher();
   if( svcDispatcher == null || svcDispatcher.getErrorCode() != 0 ||
!svcDispatcher.inqSrvcByType(CfgAppType.CFGCoBrowsingServer,
strTenant) )
   {
      //No Server is found. Load balancing is disabled.
      CoBrowseServerHost = null;
   }
   else{
      //Cobrowse server found
      CoBrowseServerHost = svcDispatcher.getSrvcHost();
   }
}catch(Exception ex)
{
   //Error load balancing is disabled.
   CoBrowseServerHost = null;
}
```

Drawing the Form

Next, this HTML code block draws a page, defines a JavaScript function that reports the Co-Browse Server host, and defines links to perform basic co-browse functions:

```
<html>
   <head>
      <link rel="stylesheet" href="/WebAPISamples761/icc_style.css"
         type="text/css"/>
      <title>MCR 7.6.1 samples. Cobrowse "Meet Me" service</title>
      ...

   <body onLoad="javascript:window_onload();"
         onunload= "javascript:window_onunload ();"
         background="/WebAPISamples761/fon.gif">
      <script LANGUAGE=javascript type="text/javascript" >
      var CobrowseHostName = null;
      <%
      if(CoBrowseServerHost != null)
      {
```

```
    out.println("CobrowseHostName = \"" + CoBrowseServerHost + "\";");
  }else{
    out.println("CobrowseHostName = ConavigationServerUrl;");
...
</script>

<h2 align="center">This sample demonstrates basic Cobrowse
    functionality</h2>
...
        <table border="1">
          <tr>
            <td colspan="2">
              <textarea style="width:600px" rows="10" id="Messages"
                NAME="Messages" READONLY></textarea>
            </td>
          </tr>
          <tr>
            <td colspan="2">
              Co-browse server host is :
              <script LANGUAGE=javascript type="text/javascript">
                document.write(CobrowseHostName);</script>
            </td>
          </tr>
          <tr>
            <td width="50%">
              <a href="javascript:CoBrowse_onclick();" >
                Start Cobrowse</a>
            </td>
            <td width="50%">
              <a href="javascript:EndCoBrowse_onclick();" >
                End Cobrowse</a>
```

Declarations

Next, this block of JavaScript declares and initializes core variables and functions. The `window_onload()` and `window_onunload()` functions are declared but 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()
{
}

function window_onunload ()
{
}
```

The AddMessage() function is declared here, and is used throughout "Event Handlers" on page 141 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, then calls two functions: HBInitializeAPI() to initialize the Co-Browse API, and HBLoginGuest() to properly log in the user. These functions are defined in the hbmessaging.js file. For details about their internals, refer to the KANA Response Live documentation listed in "Related Resources" on page 17.

```
//------ All about Cobrowse ---------------------
function CoBrowse_onclick()
{
   if(UserLoggedIn == true)
   {
     alert("Please log out user "+HBUserID+" before starting a new
        co-browse 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, and also in the following calls to the external HBLogout() function. HBLogout() is another function 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 co-browse 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 listed in “Related Resources” on page 17.

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
      }
      </script>
   </body>
</html>
```

Chat and Co-Browse Sample

This section details the sample code that is unique to the Chat and Co-Browse Sample.

Purpose

The Chat and Co-Browse Sample demonstrates how to use co-browsing functionality during a chat session.

Files

The Chat and Co-Browse Sample resides in the .../`ChatAndCoBroswe` directory. The sample consists of 11 files. Seven of these are common to all the co-browse samples, and are described in "Co-Browse Samples Overview" on page 137:

- `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 107:

- `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 co-browsing functionality. See the detailed code explanation in the next section below.

The last file is this sample's main frameset:

- `ChatAndCoBrowse.htm`

Code Explanation

The HtmlChatPanel.jsp File Explained

This section highlights ways in which the Chat and Co-Browse 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 107. The two files share certain chat functions, plus the HTML code that draws a chat form.
- The `CoBrowseEventHandler.jsp` file in "Co-Browse Sample" on page 137. This sample's `HtmlChatPanel.jsp` file adds co-browsing functions and event handlers from that file. But some event handlers differ, in ways that this section also identifies.

First, comparing this sample's `HtmlChatPanel.jsp` file to the Chat Sample's `HtmlChatPanel.jsp` file: This sample's file imports two packages that are instead handled by the Chat Sample's `HtmlChatCommand.jsp` file:

```
<%@page import="Genesys.webapi.system.loadbalancing.*"%>
<%@page import="Genesys.CfgLib.*"%>
```

Getting Load Balancer and Co-Browse Server Instances

The following `try/catch` block is also absent from the Chat Sample's `HtmlChatPanel.jsp` file. It corresponds instead to the Co-Browse Sample's `CoBrowseEventHandler.jsp` file. As in that sample, it creates a load-balancer instance and attempts to discover a Co-Browse Server host:

```
try {
   SvcDispatcher svcDispatcher = new SvcDispatcher();
   if (svcDispatcher == null || svcDispatcher.getErrorCode() != 0
    || !svcDispatcher.inqSrvcByType(CfgAppType.CFGCoBrowsingServer,
    strTenant))
    {
      //No Server is found. Load balancing is disabled.
      CoBrowseServerHost = null;
    }
   else {
      //Cobrowse server found
      CoBrowseServerHost = svcDispatcher.getSrvcHost();
    }
}
catch (Exception ex) {
   //Error load balancing is disabled.
   CoBrowseServerHost = null;
}
%>
```

Declarations

The HtmlChatPanel.jsp file's body section omits the corresponding Chat Sample file's reference to Commlib.js. However, it declares and initializes some extra variables to support co-browsing functions. Again, these declarations match those in the Co-Browse Sample's CoBrowseEventHandler.jsp file:

```
var HBApiWindow= null;
    var HBUserID= "";
    var IsFirstConavigation= true;

    var UserLoggedIn= false;
    var ConnectTo = "";
    var StartPage= "http://www.google.com";
```

Functions and Event Handlers

Like the Chat Sample's HtmlChatPanel.jsp file, this file includes a show_message() function. Elsewhere in this file, co-browsing event handlers will call show_message() to write messages to the chat frame:

```
function show_message(strMessage)
{
  document.forms[0].transcript.value =
    document.forms[0].transcript.value + "\r\n" + strMessage;
  document.forms[0].transcript.scrollTop =
    document.forms[0].transcript.scrollHeight;
}
```

Like the Co-Browse Sample's CoBrowseEventHandler.jsp file, this version of HtmlChatPanel.jsp also includes additional logic, functions, and event handlers to support interactions with the Co-Browsing Server. You will find these code blocks under the comment reproduced just below:

```
//------ All about Cobrowse ---------------------

    var CobrowseHostName = null;
    <%
    if(CoBrowseServerHost != null)
    {
      out.println("CobrowseHostName = \"" +
        CoBrowseServerHost + "\";");
    }else{
      out.println("CobrowseHostName = ConavigationServerUrl;");
    }
    %>

    function CoBrowse_onclick()
    {
            if(UserLoggedIn == true)
            {
               alert("Please log out user "+HBUserID+" before starting
                  a new co-browse session.");
            }else{
...
```

The first three event handlers are similar to identically-named event handlers in the Co-Browse Sample's `CoBrowseEventHandler.jsp` file. But where that file's event handlers call an `AddMessage()` function, this file's event handlers instead call its `show_message()` function (see "Functions and Event Handlers" on page 145) so as to write their messages to the chat frame:

```
/**** HB API Events handlers ****/
      function HBCouldNotConnect( sReasonID )
      {
         show_message("Can't connect to Cobrowse server.
            Reason: "+sReasonID);
      }

      function HBLoginError(reasonID, description)
      {
         show_message("Can't login to Cobrowse server. Reason: " +
            reasonID + " Description: " + description);
      }

      function HBJoinedSuccessfully()
      {
         IsAgentJoined = true;
         show_message ("Agent joined.");
      }
```

This file's `HBJoinRequested()` event handler differs in the same way from that in the Co-Browse Sample: Rather than calling the `AddMessage()`, it calls `show_message()` to identify the agent to the caller:

```
      function HBJoinRequested(sName)
      {
         show_message ("Your agent is : " + sName + ".");
         return true;
      }
```

This file's `HBLoggedIn()` event handler differs in two ways from its counterpart in the Co-Browse Sample. First, it calls `show_message()` instead of `AddMessage()`. Second, upon verifying the customer's login and connection, it writes the customer's co-browse ID to the chat frame before calling `HBCreateSession()`:

```
      function HBLoggedIn(sHipboneID)
```

```
    {
       HBUserID = sHipboneID;
       show_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 co-browsing with the customer:

```
       if (bConnected == true)
       {
          document.forms[0].message.value =
             "$$CUSTOMER_COBROWSING_ID=" + HBUserID;
          on_send();
       }

       HBApiWindow.HBCreateSession();
    }
```

Finally, this file's remaining event handlers each differ from their Co-Browse Sample counterparts by internally calling `show_message()` instead of `AddMessage()`:

- `HBLoggedOut()`
- `HBSessionStarted()`
- `HBLinkConavigated()`
- `HBUserExitedSession()`
- `HBUserEnteredSession()`

Drawing the Chat Form and Co-Browse Links

Along with the HTML code that creates the traditional chat form, the code also displays the Co-Browsing Server host's name, and displays links to start and stop a co-browse session:

```
<td colspan="6">
   Co-browse server host is :
   <script LANGUAGE=javascript type="text/javascript">
   document.write(CobrowseHostName);</script>
</td>
...
<td colspan=3 align="center">
   <a href="javascript:CoBrowse_onclick();">Start Cobrowse</a>
   &nbsp;&nbsp;&nbsp;
   <a href="javascript:EndCoBrowse_onclick();">Stop Cobrowse</a>
</td>
```

Co-Browse Meet Me

The section details the sample code that is unique to the Co-Browse Meet Me Sample.

Purpose

The Co-Browse Meet Me Sample code demonstrates how to set up a co-browse session with a specific other person whose co-browse ID the user knows.

Files

The .../CoBrosweMeetMe directory contains nine files; eight of these are documented in “Co-Browse Samples Overview” on page 137. This section focuses on code unique to this sample’s CoBrowseEventHandler.jsp file (as compared to the corresponding file in “Co-Browse Sample” on page 137).

The CoBrowseEventHandler.jsp File Explained

Entering the Other Party’s Co-Browse ID

Compared to the Co-Browse Sample’s CoBrowseEventHandler.jsp file, this sample adds an input box and a link by which the user can specify the co-browse ID of the person he or she wants to browse with:

```
<tr>
   <td colspan="2">
      Connect to:
      <input type="text" ID=ConnectTo value="" name="ConnectTo"/>
   </td>
   </tr>
   <tr>
      <td width="50%">
            <a href="javascript:CoBrowse_onclick();" >
               Cobrowse with other person</a>
      </td>
```

“Meeting” the Other Party

The above code captures a ConnectTo value, which represents the target co-browse ID. Below, this sample’s CoBrowse_onclick() function includes some additional logic that attempts to connect to this ID, and notifies the user if the ConnectTo value is empty:

```
function CoBrowse_onclick()
```

```
{

   ConnectTo = 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");
...
```

Event Handlers

Most of this file's event handlers correspond to those in the Co-Browse Sample's `CoBrowseEventHandler.jsp` file. But there are some minor differences.

This file's `HBLoggedIn()` event handler omits the internal call to `HBApiWindow.HBCreateSession()` found in the Co-Browse Sample's `HBLoggedIn()` event handler, because we can assume that the other party has already created the session:

```
function HBLoggedIn(sHipboneID)
{
   HBUserID = sHipboneID;
   AddMessage ("CobrowseID : " + sHipboneID + ".");
   UserLoggedIn = true;
}
```

Similarly, this file's `HBSessionStarted()` event handler differs from its Co-Browse Sample counterpart by omitting an internal call to `HBApiWindow.HBConavigateLink(StartPage, null)`. Again, we can assume that the other party has already established the URL to co-browse:

```
function HBSessionStarted()
{
   AddMessage ("HBSessionStarted()");
}
```

For similar reasons, this file's `HBUserEnteredSession()` event handler omits its Co-Browse 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
}
```

Co-Browse Init Start Page

This section details the sample code that is unique to the Co-Browse Init Start Page Sample.

Purpose

The Co-Browse Init Start Page Sample code demonstrates basic co-browsing with a specified initial start page.

Files

This section focuses on code that is unique to this sample, as compared to the basic "Co-Browse Sample" on page 137. The .../CoBrosweInitStartPage directory contains nine files; seven of these are common files already described in "Co-Browse Samples Overview" on page 137. Two files differentiate this sample:

- InitialStartPageExample.html—Replaces the basic Co-Browse Sample's CoBrowse.htm file, but serves a similar function: sets up the basic display frame as a container for imported logic.
- CoBrowseEventHandler.jsp—Contains some additional logic, compared to the Co-Browse Sample's file of the same name. For details, see the following section.

The CoBrowseEventHandler.jsp File Explained

Specifying the Start Page

Compared to its counterpart in the Co-Browse Sample, this sample's CoBrowseEventHandler.jsp file contains one additional code block that:

- Provides an input box to specify the other party's co-browse ID.
- Provides an input box to confirm or override the initial URL that both users will co-browse
- Provides a link to open that page.

Here is the added code:

```
<h2 align="center">MCR 7.6.1 samples. Cobrowse start page
sample.</h2>
...
   Connect to:
   <input type="text" ID=ConnectTo value="" name="ConnectTo"/>
</td>
<td>
   Initial Start Page :
```

```
   <input type="text" ID=StartPage value="http://www.yahoo.com" 
name="StartPage"/>
</td>
...
<td>
   <a href="javascript:CoBrowse_onclick();" >Cobrowse with other 
person</a>
</td>
```

Event Handlers

Most of this file's event handlers correspond to those in the Co-Browse Sample's `CoBrowseEventHandler.jsp` file. This section identifies the minor differences.

In this file's version of the `HBLoggedIn()` event handler, the internal call to `HBApiWindow.HBCreateSession()` is embedded in an `if` branch. This `if` branch verifies a connection to the other party before creating a session:

```
function HBLoggedIn(sHipboneID)
{
   HBUserID = sHipboneID;
   AddMessage ("CobrowseID : " + sHipboneID + ".");

   UserLoggedIn = true;

   if(ConnectTo == "")
   {
      HBApiWindow.HBCreateSession();
   }
}
```

This file's `HBSessionStarted()` event handler differs from its Co-Browse Sample counterpart by omitting an internal call to `HBApiWindow.HBConavigateLink(StartPage, null)`. We assume that the other party has already established a co-browse session and established the URL to co-browse:

```
function HBSessionStarted()
{
   AddMessage ("HBSessionStarted()");
}
```

That `HBApiWindow.HBConavigateLink(StartPage, null)` call instead occurs in this file's `HBUserEnteredSession()` event handler. (The Co-Browse Sample's version of that event handler makes no such call). The call is embedded in an `if` branch that first verifies the other party's co-browse ID. It co-navigates 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);
   }
}
```

Co-Browse Dynamic Start Page

The section describes the Co-Browse Dynamic Start Page Sample's code.

Purpose

The Co-Browse Dynamic Start Page Sample demonstrates how one party can click a co-browse control on an HTML page containing a form. When co-browse begins, all the form data already entered is dynamically prefilled in the other party's co-browse window. So a customer and agent can dynamically access each other's form data.

Files

The .../`CoBrosweDynamicStartPage` directory contains five files. For the sample to work properly, place all the files together into a public web directory. Four of these files (listed in "Co-Browse Dynamic Start Page" on page 59) should not be modified.

This section focuses on the file that you can modify—the sample's main file, `ExampleOfDynamicStartPage.jsp`. This file serves some of the same purposes as the `CoBrowseEventHandler.jsp` file in "Co-Browse Sample" on page 137; therefore, this discussion focuses on the unique code that it provides.

The co-browse session starts from the `ExampleOfDynamicStartPage.jsp` page, where it collects user input. To check the sample, enter arbitrary information into this page's form fields, then click the `Live Help` link. You will see a KANA co-browse window specific to the dynamic start page.

Code Explanation

This sample's `ExampleOfDynamicStartPage.jsp` file begins with statements similar to those in the samples described above: `import` and `include` statements, and a `try-catch` block that seeks to obtain a load balancer instance.

Next, the `<html>` section calls the file's supporting stylesheet, JavaScript, and constants files:

```
<html>
   <head>
      <link rel="stylesheet" href="/WebAPISamples761/icc_style.css"
         type="text/css"/>
      ...
      <script LANGUAGE=javascript type="text/javascript"
         SRC="responseLive.js"></script>
      <script LANGUAGE=javascript type="text/javascript"
         src="/WebAPISamples761/Constants.js"></script>
   </head>
```

Next, the sample calls its `window_onload()` function (defined later in the file), then reports any discovered Co-Browse Server host:

```
<body onload="return window_onload();"
   background="/WebAPISamples761/fon.gif">
      <script type="text/javascript">
      var CobrowseHostName = null;

      <%
      if(CoBrowseServerHost != null)
      {
         out.println("CobrowseHostName = \"" + CoBrowseServerHost +
            "\";");
      }else{
         out.println("CobrowseHostName = ConavigationServerUrl;");
      }
      %>
```

The code next defines an `openLiveHelp()` function. This function opens the Dynamic Start Page API window when the user clicks the `Live Help` link:

```
function openLiveHelp()
      {
         var customerName = window.document.forms[0].fullName.value;
         startDSPMeetMe(ConavigationiChannelID, null, customerName,
            null, CobrowseHostName)
      }
```

Next, the code defines a link to the `Live Help` functionality, which is provided by the other files in the sample's installed directory:

```
<a href="javascript:openLiveHelp()">Live Help</a>
```

The remainder of the file builds the form that solicits the user's input of personal information. Two text boxes prompt for the user's name and e-mail address:

```
<form name="checkout_form" action="" method="post">
   <table width="760" border="0" cellspacing="0" cellpadding="0">
      <tr>
         <td valign="middle" align="left" width="525">Full Name</td>
         <td>
            <input type=text class="body" style="width: 215px"
               size="18" maxlength="16" value="" name="fullName"/>
         </td>
      </tr>
      <tr>
         <td valign="middle" align="left" width="525">
            E-Mail Address</td>
         <td>
            <input type=text class="body" style="width: 215px"
               size="18" maxlength="16" value=""
               name="emailAddress"/>
         </td>
      </tr>
```

A very basic drop-down list offers three `State/Province` options; then another text box prompts for the user's password:

```
<td>
   <select name="state">
      <option value="">State/Province</option>
      <option value="CA">CA</option>
      <option value="NY">NY</option>
      <option value="TX">TX</option>
   </select>
...
<td>
   <input type="password" style="width: 215px" size="18"
      maxlength="16" value="" name="password"/>
</td>
```

Next, a radio-button panel prompts the user to select a credit-card type:

```
            <tr>
               <td valign="middle" align="left" width="525">
                  Credit Card Type</td>
               <td>
                  <input type="radio" value="Visa" checked
name="creditCardType"/>
                  Visa
                  <input type="radio" value="MasterCard"
name="creditCardType"/>
                  Master Card
            <input type="radio" value="AmericanExpress"
name="creditCardType"/>
            American Express
            <input type="radio" value="Discover"
name="creditCardType"/>
            Discover
          </td>
        </tr>
        <tr>
```

Finally, a multiple-selection box allows the user to specify one or more preferred callback times:

```
<td valign="middle" align="left" width="525">Preferred callback
time</td>
<td>
  <select multiple name=selectbox1 size=3>
    <option value="Morning">Morning</option>
    <option value="Day time">Day time</option>
    <option value="Evening">Evening</option>
  </select>
...
```

Build Your Own Dynamic Start Page Example

To add dynamic start page functionality to your own custom page, first import these libraries:

```
<%@ page import="Genesys.webapi.system.loadbalancing.*" %>
<%@ page import="Genesys.CfgLib.*" %>
```

Next, load this JavaScript file that provides the co-browse API:

```
<script LANGUAGE=javascript type="text/javascript"
  SRC="responseLive.js"></script>
```

Then insert the following script:

```
<%
String  CoBrowseServerHost = null;

try{
  SvcDispatcher svcDispatcher = new SvcDispatcher();
  if( svcDispatcher == null || svcDispatcher.getErrorCode() != 0 ||
!svcDispatcher.inqSrvcByType(CfgAppType.CFGCoBrowsingServer,
strTenant) )
  {
    //No Server is found. Load balancing is disabled.
    CoBrowseServerHost = null;
```

```
   }else{
      //Cobrowse server found
      CoBrowseServerHost = svcDispatcher.getSrvcHost();
   }
}catch(Exception ex)
{
   //Error load balancing is disabled.
   CoBrowseServerHost = null;
}
%>

...
      <script type="text/javascript">
      var CobrowseHostName = "<your_co-browse_server_host-name>";

      <%
      if(CoBrowseServerHost != null)
      {
         out.println("CobrowseHostName = \"" + CoBrowseServerHost +
"\";");
      %>
...
      function openLiveHelp()
      {
         startDSPMeetMe(ConavigationiChannelID,null,"guest",null,
            CobrowseHostName);
      }
```

To each page that will incorporate dynamic start page functionality, add a reference that starts the `openLiveHelp()` function :

```
<a href="javascript:openLiveHelp()">Live Help</a>
```

Settings

To ensure that your dynamic start page application will work, observe these 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 Co-Browse Server's `.../hbroot/certs/` folder.
- The dynamic start 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 of whether cookies can be set in the user's browser.

Limitations of the Dynamic Start Page

- **IP Checks**—The dynamic start page does not work if your web site associates identification and cookies with the IP address of the customer's computer. 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 start page to re-appear after a POST submission.
- **Cookies**—The dynamic start page feature only traps the cookies that are accessible from the domain and path at which the `Help` button is located.
- **Large Cookies**—The dynamic start page feature cannot support cookies greater than 1K in size.
- **Frames in Different Domains**—Pages with frames with different domains or protocols must be supported on a custom basis.
- **Start Pages within Frames**—Not supported.
- **Multiple Submissions**—If your web site does not reload URLs, the start page feature may not function properly.

FAQ

The section explains the FAQ Sample’s code.

Purpose

The FAQ Sample code demonstrates a basic FAQ Service.

Files

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

Code Explanation

The first line in the `FAQ.jsp` file sets the page content type:

```
<%@ page contentType = "text/html; charset="windows-1252" %>
```

The second line is a call to the `response.setContentType()` method. This sets the content type or character encoding to use in the response to the client:

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

Then the sample code loads the following libraries into memory:

```
<%@ page import="Genesys.webapi.media.faq.direct.*" %>
<%@ page import="java.io.*" %>
<%@ page import="java.util.*" %>
<%@ page import="java.text.*" %>
<%@ page import="Genesys.webapi.utils.i18n.*" %>
```

The file contains six functions:

- `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 using the list of all categories.
- `getCategory()`—Gets a category using the list of all categories.
- `getByName()`—Gets a child from of the Root Category by name.

The remaining code creates a form with a drop down list of categories to choose from and a button to retrieve the FAQs related to the selected category. The form also provides an input box for your direct questions, and a button to submit your questions. This button will retrieve all the FAQs in the selected category that relate to your question. In addition to the list of FAQs, it will also provide a number that represents the confidence score (ranging from 0 to 100) that the FAQ relates to your direct question. Once you make a selection from this list, you are presented with the answer, and also a group of radio buttons that you can used to provide feedback about the accuracy of the answer:

```
...
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();

   String id= i18nsupport.GetSubmitParametr(request, "id");
   String typeOfRequest= i18nsupport.GetSubmitParametr(request,
"submit_type");
   String ctgName= i18nsupport.GetSubmitParametr(request, "ctg");
   String question= i18nsupport.GetSubmitParametr(request,
"question");
   String accuracy= i18nsupport.GetSubmitParametr(request,
"accuracy");

   if (accuracy != null && accuracy.equals("") == false)
   {
      _faq_category ctg = getCategory(id);
      double ac = Double.parseDouble(accuracy);
```

This 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 (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("")))
{
   %>
   <h2 align="center">Web Self Service example</h2>
   <FORM method="post" action="FAQ.jsp" >
   <hr width="100%" align="center">
   <h2>Select the Scope</h2>
   <SELECT name="ctg">
     <OPTION selected>All Categories
     <%
        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());
        }
     %>
   </SELECT>

   <h2>Get the List of All FAQ</h2>

 <br>
 <INPUT type= "submit" value="Get FAQ" name="submit_type">
 <%
     if (wss.is_model_available())
     {
 %>
     <hr width="100%" align="center">
     <h2>Get the Answer to the Question</h2>

     <br>
     <TEXTAREA  name="question" rows="3" cols="40"></TEXTAREA>
     <br>
     INPUT type= "submit" value="Get Answer" name="submit_type">
   </FORM>
 <%
```

```
      }
         }else if (id != null && id.equals("") == false){
            _faq_response r = getResponse(id);
    %>
         <h1> Text of the Answer </h1>
    <%
         if (r != null)
            out.println(r.get_text());
            out.println("<br><br><br>
            <a href=\"javascript:window.history.back();\">
               <b>Return<b></a>");

         if (question != null && question.equals ("") == false)
         {
            out.println("<BR><BR>Original question:<BR>");
            out.println("<FORM method=\"post\" action=\"FAQ.jsp\">");
            out.println("<TEXTAREA readonly name=\"question\"
rows=\"3\"
               cols=\"40\">"+question+"</TEXTAREA>");
            out.println("<INPUT type= \"hidden\" value=\"" + id + "\"
               name=\"id\">");
   %>
```

This code displayes the accuracy radio buttons:

```
<BR><BR>
Please rate the accuracy level of the answer for your question:
<BR>
<input type="radio" value="-1" name="accuracy"
   onclick="document.forms[0].submit();">-10
<input type="radio" value="-0.5" name="accuracy"
   onclick="document.forms[0].submit();">-5
<input type="radio" value="-0" name="accuracy"
   onclick="document.forms[0].submit();">0
<input type="radio" value="0.5" name="accuracy"
   onclick="document.forms[0].submit();">5
<input type="radio" value="1" name="accuracy"
   onclick="document.forms[0].submit();">10
<%
   out.println("</FORM>");
}

}else
{
   if(question == null)
      question = "";

      current_ctg = getByName(ctgName);
      // Request for classification
      if (("Get Answer".equalsIgnoreCase(typeOfRequest)) &&
          (question.trim().length()>0) )
    {
      DecimalFormat format = new DecimalFormat("00.");
      out.println("<h1>Answers List</h1>");
      List faq = wss.get_classification_list(current_ctg,
        question, 0.1);
      out.println("<ol>");
      for (Iterator i = faq.iterator(); i.hasNext(); )
      {
        _faq_result item = (_faq_result) i.next();
        out.println("<LI>
          <a href=\"javascript:document.forms[0].id.value='"+
            ctgList.indexOf(item.get_category())+
            "';document.forms[0].submit();\">" +
            getCategoryDescription(item.get_category()) + " ("+
            format.format(item.get_confidence_level()) + ") </
          a>");
}
```

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, we have to pass the original question in either a hidden form field or a read-only text area. This complicates our example slightly. If you do not want to implement the feedback feature, simply 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:

```
out.println("</ol>");
out.println("<a href=\"FAQ.jsp\"><b>Return<b></a>");
out.println("<BR><BR>Original question:<BR>");
out.println("<FORM method=\"post\" action=\"FAQ.jsp\">");
out.println("<TEXTAREA readonly name=\"question\" rows=\"3\"
  cols=\"40\">"+question+"</TEXTAREA>");
  out.println("<INPUT type= \"hidden\" value=\"\" name=\"id\">");
  out.println("</FORM>");
}else
{ // Request for FAQ List
  out.println("<h1>FAQ List</h1>");
  List faq = wss.get_faq_list(current_ctg, 0);
  out.println("<ol>");
  for (Iterator i = faq.iterator(); i.hasNext(); )
  {
    _faq_result item = (_faq_result) i.next();
    out.println("<LI><a href=\"FAQ.jsp?id=" +
    ctgList.indexOf(item.get_category()) + "\">" +
    getCategoryDescription(item.get_category()) + " (" +
    item.get_frequency() + ") </a>");
  }
  out.println("</ol>");
  out.println("<a href=\"FAQ.jsp\"><b>Return<b></a>");
}......
```

Open 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 lets you enter data to be attached to the interaction, such as first and last name, and three key-value pairs. Fields such as host, port ID, and workflow queue are automatically filled into the form for you. The information for these fields is provided by 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 Open Media Sample is a JSP file that shows how to:

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

Functionality Overview

The following sections review the code used in implementing the different Open Media functions:

- "Setting the Content Type and Character Encoding" on page 163
- "Loading Libraries and Importing Files" on page 163
- "Handling Content" on page 163

Files

The .../`ItxSubmit` directory contains the Open Media Sample. The sample consists of a single file, `ItxSubmit.jsp`.

Code Explanation

The following subsections explain the code in ItxSubmit.jsp. 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 most important points. In these cases, the missing lines have been replaced with "...".

Setting the Content Type and Character Encoding

The first line 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 Universal Callback Server Application object in Configuration Server:

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

Loading Libraries and Importing Files

Then the sample code loads the following libraries into memory:

```
<%@ page import="Genesys.webapi.system.loadbalancing.*" %>
<%@ page import="Genesys.webapi.utils.i18n.*" %>
<%@ page import="Genesys.webapi.media.common.*" %>
<%@ page import="Genesys.webapi.media.interaction.direct.*" %>
<%@ page import=
  "Genesys._workflow_engine.protocols._workflow_engine_protocol.*"
%>
<%@ page import="Genesys._workflow_engine.protocols.common.*" %>
<%@ page import="com.genesyslab.list.*" %>
<%@ page import="java.io.*" %>
<%@ page import="java.net.*" %>
<%@ page import="java.lang.System" %>
```

The code also needs to access the constants.jsp file:

```
<%@ include file="../constants.jsp" %>
```

Handling Content

Creating the HTML Header

Next the code writes some HTML header tags:

```
<html>
<head>
```

```
<link rel="stylesheet href="/WebAPISamples761/icc_style.css"
type="text/css">
<title>MCR Samples 7.6.1 Interactions Server</title>
</head>
<body LANGUAGE=javascript onload="return window_onload();"
     background="/WebAPISamples761/fon.gif">
```

Retrieving Parameters

Now the Java code section retrieves the request parameters, such as the host and port name, first and 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.

```
<%
    String strError = "";
    boolean bError = false;
    int rc = 0;

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

    String strPort = request.getParameter("port");
    if( strPort == null ||  strPort.equals("") ) strPort = "";
 ...
```

Creating a New Instance

This section of code declares a new instance of `_interaction_direct` for communicating with Interaction Server. If the user has chosen an action, the code then instantiates the new instance using the host and port IDs of Interaction Server. It also registers the user with the Interaction Server. If there are any errors or exceptions, the code now handles them.

```
_interaction_direct itx = null;
   if (strAction != null && !strAction.equals(""))
    {
      try
      {
        itx = new _interaction_direct (strHost,
           Integer.parseInt(strPort));
        rc = itx.register ("JSPSample");
        if (rc != _interaction_direct.__rc_ok)
        {
          strError = itx.lasterror();
          bError = true;
        }
```

```
          }
          catch (_communication_exception iex)
          {
            strError = iex.toString();
            bError = true;
          }
          catch (_we_pack_exception iex)
          {
            strError = iex.toString();
            bError = true;
          }
        }
```

Getting Load Balancer and Processing Your Request

This section of code checks to see whether you have selected the `Send` action. If so, it collects the parameter data sent from the browser into a `TKVList` for transmission to Interaction Server. Then it creates a load balancer instance and submits the data. In case the server generated the interaction ID, the code sets its interaction ID variable to the value stored on the server. If the interaction ID is not generated by the server you must ensure its uniqueness. After the action has been processed, the load balancer instance is set to `null` to avoid leaving an orphaned connection.

```
if (strAction != null && strAction.equals("Send") && bError == false)
    {
      try
      {
        com.genesyslab.list.TKVList user_data = new com.genesyslab.list.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);
        if (bAgree)
          user_data.addInt ("Agree", 1);
        else
          user_data.addInt ("Agree", 0);
        user_data.addInt ("Gender", iGender);

   SvcDispatchersvcDispatcher= new SvcDispatcher();
   Long  longTenantID= svcDispatcher.getTenantId(strTenant);
   int   iTenantID= -1;

   if (longTenantID != null)
     iTenantID = longTenantID.intValue();

        rc = itx.submit(strMediaType, strIteractionID, strScriptName, iTenantID,
   user_data);
```

```
        if (rc != _interaction_direct.__rc_ok)
        {
          strError = itx.lasterror();
          bError = true;
        }
   //If IteractionID was generated by server.
   strIteractionID = itx.get_interaction_id();
   ...
   svcDispatcher= null;
}
```

If you have selected `Update`, the JSP will delete the third key-value pair from the interaction that you specify, using the `change_properties` method.

```
else if (strAction != null && strAction.equals("Update") && bError == false)
    {
      try
      {
        com.genesyslab.list.TKVList deleted_user_data = new
   com.genesyslab.list.TKVList();//Remove only one KV pair
        deleted_user_data.addString (attachDataName3, attachDataValue3);

        com.genesyslab.list.TKVList user_data = new com.genesyslab.list.TKVList();
        user_data.addString (attachDataName1, attachDataValue1);
        user_data.addString (attachDataName2, attachDataValue2);
        user_data.addString ("FirstName", strFirstName);
        user_data.addString ("LastName", strLastName);
        if (bAgree)
          user_data.addInt ("Agree", 1);
        else
          user_data.addInt ("Agree", 0);
        user_data.addInt ("Gender", iGender);

        rc = itx.change_properties(strIteractionID, user_data, deleted_user_data);
       ...
```

If you have chosen to cancel the interaction, your instance of `_interaction_direct` will issue the `stop_processing` method, which cancels the interaction.

```
else if (strAction != null && strAction.equals("Cancel") && bError == false)
    {
      try
      {
        rc = itx.stop_processing(strIteractionID, 12345, "Put your reason here");
      ...
```

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.

```
    try
    {
      if (itx != null)
        itx.close();
      itx = null;
    }
    catch (_communication_exception iex)
    {
      strError = iex.toString();
      bError = true;
    }

    if (bError == false && strAction != null && !strAction.equals(""))
      strError = "Interaction succesfully submitted to Interaction server. Please check
        Interaction Server log for details.";
%>
```

Constructing the HTML Body

Next, the code includes the HTML code to create the input boxes, the `Send`, `Update`, and `Cancel` buttons, and the field that holds messages from the server:

```
<form method="post" action="ItxSubmit.jsp">
      <H2 align="center">MCR 7.6.1 samples. Custom Web Form Submit.</H2><BR>
<table border="1">
<tr>
    <td colspan=2 align="center"><H2>Enter information for Interaction Server</H2></td>
</tr>
<tr>
    <td>Interaction Server host</td>
    <td><INPUT TYPE="String" NAME="host" value="<%=strHost%>"></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`.

Chapter

10 Multimedia Simple Samples for .NET

This chapter examines Genesys Multimedia's simple, web-based samples for .NET, and their code. (Java developers should instead see Chapter 9, "Multimedia Simple Samples for Java," on page 95.) This chapter covers the following topics:

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 voice callback, chat, e-mail, statistics, history, and Open Media submission functions in your web application, by reviewing the key functions in the respective samples. The samples come with the Multimedia Interactive Management CD. For information on installing the samples, see Chapter 2, "About the Samples," on page 51.

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 Web 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 Simple Samples discussed here are:

- Web-Based Voice Callback
- Web-Based Chat
- Web-Based Chat with AJAX
- Web-Based E-mail
- Web-Based Open Media Interaction Submission
- Web-Based Stat Server
- Web-Based Universal Contact Server

Web-Based Voice Callback

This sample demonstrates how to use the callback API (see “Callback” on page 28) to implement voice callback on a web form.

Web-Based Chat

This sample demonstrates how to use the chat API for the Flex Chat protocol (see “Chat” on page 31) to implement a chat feature on a web form.

Web-Based Chat with AJAX

This sample demonstrates how to use the chat API for the Flex Chat protocol (see “Chat” on page 31) to implement a chat feature on a web form.

Web-Based E-Mail

This sample demonstrates how to use the e-mail API (see “E-Mail” on page 35) to send e-mail using a web form.

Web-Based Open Media

This sample demonstrates how to use the Open Media API (see “Open Media” on page 37) to submit and update an interaction using a web form.

Web-Based Stat Server

This sample demonstrates how to use the Stat Server API (see “Statistics Packages” on page 39) to select predefined statistics and retrieve their current value using a web form.

Web-Based Universal Contact Server

This sample demonstrates how to use the Universal Contact Server API (see “Universal Contact Server” on page 39) to access a contact’s interaction history using a web form.

Files Included: .ASPX Versus .ASPX.CS

Each Simple 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” below.
- `App_Code\Constants.cs`—see “Constants” on page 172.
- `Global.asax`—see “Startup Shutdown, and Load Balancing” on page 172.

- `icc_style.css`—see "Style Sheets" on page 173.
- `index.htm`—see "Greetings Page" on page 173.
- `ip_description.xml`—XML describing the installation package.
- `mcr_style.css`—see "Style Sheets" on page 173.
- `read_me.html`—HTML describing the installation package.
- `Web.Config`—see "Licensing, Authentication, and Error Reporting" on page 172.
- `arrow.gif`—see "Graphics" on page 173.
- `fon.gif`—see "Graphics" on page 173.
- `genesyslogo-trans.gif`—see "Graphics" on page 173.
- `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) files, `icc_style.css` and `mcr_style.css`, contain 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.

Callback Sample

This section presents the purpose, functionality overview and code implementation for the Callback Sample.

Purpose

The Callback Sample shows how to:

- Collect user data from a form submission.
- Mask potentially dangerous server-side data or symbols such as <, >, and ".
- Connect to Universal Callback Server using the Callback API.
- Submit a callback request.
- Cancel a callback request.
- Get information about a request.
- Analyze the server’s responses to requests, handling errors.
- View a list of the user’s callback requests.
- Display request results in a table.

Functionality Overview

The following sections review the code used in implementing the different callback functions:

- “Converting Time Data and Drawing the Form” on page 174
- “Declaring and Importing Packages” on page 175
- “Declaring Variables” on page 175
- “Trapping HTML Characters” on page 176
- “Retrieving Form Data” on page 176
- “Processing the Request” on page 176

Files

The ...\Callback directory contains the Callback Sample. The sample consists of two files, Callback.aspx and Callback.aspx.cs.

Code Explanation

The following subsections explain the code in the Callback.aspx and Callback.aspx.cs files. They appear in the same order as the code in the respective files. In some places, however, code snippets omit several lines of code in order to focus your attention on the most important points. In these cases, the missing lines have been replaced with “...”.

User Interface Implementation

The Callback.aspx file contains most of the code that presents information to the user.

Converting Time Data and Drawing the Form

The Callback.aspx page begins with a ConvertTime(UTCtime) function that converts UTC (Universal Time, Coordinated; formerly Greenwich Mean Time) time values into the local date and time.

Next, some HTML code defines a user input form, including drop-down lists enabling the user to select the local time and date.

Handling User Events

The Callback.aspx file also includes several JavaScript functions to ensure that mandatory fields are filled, to gather the user’s input, and to process this input:

- window_onload()—called every time the page is loaded. This function currently sets the form’s default callback time to the time the page was loaded, but you may also modify it to execute any other tasks that should be carried out whenever the page is loaded.

- `on_view_list()`—triggered by the `View list` Submit action. Returns `false` if the user has not entered a contact number, thus preventing the submission of the form.
- `on_request()`—triggered by the `Request Callback` Submit action. Returns `false` if the user has not entered a contact number, thus preventing the submission of the form. Otherwise, it generates a start and end date for the callback request.
- `GetUTCTime(date)`—converts the specified `date` to the equivalent UTC time.
- `double_digit(Value)`—ensures that the `Value` passed to the function contains two digits.
- `setSelection(objControl, Value)`—sets the selected value of a document field (`ObjControl`) to `Value`.
- `getSelectedOption(opt)`—takes a collection of options for a field and returns the option that has been selected by the user.
- `on_reset()`—resets the form to its default values.

Logic Implementation

The `Callback.aspx.cs` file contains most of the Callback Sample's logic. The subsections below explain the code in that file.

Declaring and Importing Packages

The `Callback.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.Callback;
using Genesyslab.Platform.WebMedia.Protocols.Callback.Events;
using Genesyslab.Platform.WebMedia.Protocols.Callback.Requests;
using Genesyslab.WebApi.Core.ConfigServer;
using Genesyslab.WebApi.Core;
using Genesyslab.Platform.Configuration.Protocols.ConfServer;
```

Declaring Variables

Next, the `Callback.aspx.cs` file's `Callback_Callback` class declares internal variables. These variables represent data that the user enters in the form drawn by `Callback.aspx`:

```
public partial class Callback_Callback : System.Web.UI.Page
{
    string svcHost;
    int svcPort;
    string strAction;
    string strFirstName;
    string strLastName;
    string strPhoneNumber;
    string strEmailAddress;
    string strMedia;
    MediaType mtMedia = MediaType.Voice;
    ...
```

Trapping HTML Characters

Next, the `Callback.aspx.cs` file's `mask_html()` function defines HTML-safe transformations of certain characters. Later functions in the file call this function on user-entered data.

Retrieving Form Data

The `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:

```
ClientScript.RegisterHiddenField("StartTime", "");
StartTime.Visible = false;
ClientScript.RegisterHiddenField("EndTime", "");
EndTime.Visible = false;
...
```

Next, it defines a query-string collection, and collects user data from the submitted form into the variables declared above:

```
NameValueCollection qs      = Request.QueryString;
strAction                   = cmd.Text;
strFirstName                = FirstName.Text;
strLastName                 = LastName.Text;
strPhoneNumber              = PhoneNumber.Text;
strEmailAddress             = EmailAddress.Text;
strMedia                    = Media.Text;
...
```

This statement searches the query string for an `strAction` (command) value, if that information was not posted with the form data:

```
if (strAction == "")
   strAction = qs.Get("cmd"); //try to get it from Query String
```

Processing the Request

The following statement accesses global constants:

```
SimpleSamplesConstants ssc  = new SimpleSamplesConstants();
```

This branch tests for an empty `strAlias`, as an indicator that no Universal Callback Server instance is already dedicated to the pending request:

```
if (strAlias == "")
{
```

Getting Load Balancer and Universal Callback Server Instance

The next code block creates a load-balancer instance, and seeks a new instance of Universal Callback Server for this request.

```
try
{
   ServiceInfo si = LoadBalancer.GetServiceInfo
      (CfgAppType.CFGUniversalCallbackServer, ssc.TenantName);
   svcHost            = si.Host;
   svcPort            = si.Port;
   bServiceAvailable  = true;
   strAlias           = si.Alias;
}
catch (LoadBalancerException e1)
{
   strErrorMessage +=
      "Callback service is not available at this time.
      lease try it later.";
}...
```

This alternative branch seeks to reuse an existing Universal Callback Server instance that has already been dispatched:

```
else
{
   try
   {
      ServiceInfo si = LoadBalancer.GetServiceInfo(strAlias);
      svcHost            = si.Host;
      svcPort            = si.Port;
      bServiceAvailable  = true;
   }
   catch (LoadBalancerException e1)
   {
      strErrorMessage +=
         "Callback service is not available at this time.
         Please try it later.";
   }
}
```

Preparing Classes for Requests

This code block prepares classes for request submissions:

```
if (bServiceAvailable)
{
   IMessage imResponse = null;
   Uri callbackServerURI = new
      Uri("tcp://" + svcHost + ":" + svcPort.ToString());
   Endpoint callbackEndPoint = new Endpoint(callbackServerURI);
   Genesyslab.Platform.WebMedia.Protocols.CallbackProtocol
      callback = new CallbackProtocol(callbackEndPoint);
   //callback.EnableLogging(new TraceLogger());
   callback.Open();...
```

Submitting Requests

The remaining body of the `Callback.aspx.cs` file consists of several branches like the snippet below. Each branch attempts to execute a specified request by:

1. Filling a collection with user-entered data.
2. Submitting the resulting request to the Universal Callback Server.
3. Analyzing the response.
4. Relaying any returned error messages.

```
if (strAction == "Request callback")
{
   KeyValueCollection userdata = new KeyValueCollection();
   userdata.Add("FirstName", strFirstName);
   userdata.Add("LastName", strLastName);
   userdata.Add("EmailAddress", strEmailAddress);

   RequestCallback reqCallback =
      RequestCallback.Create(strPhoneNumber,
         mtMedia, false, strStartTime, strEndTime, "new", userdata);
   imResponse = callback.Request(reqCallback,
         new TimeSpan(0, 0, 30));
   if (imResponse != null && imResponse.Name
         == EventAck.MessageName)
   {
      EventAck eventAck = imResponse as EventAck;
      strRequestId = eventAck.RequestId;
      strAction = "View list";
   }
   else if (imResponse != null && imResponse.Name ==
               EventError.MessageName)
   {
      EventError eventError = imResponse as EventError;
   }
}
```

Cleaning Up

The `Callback.aspx.cs` file ends by disconnecting from the Universal Callback Server—an important step to avoid leaking system resources:

```
callback.Close();
```

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 182
- "Declaring Variables" on page 183
- "Retrieving Form Data" on page 183
- "Connecting to Chat Server" on page 184
- "Creating a Chat Session" on page 185
- "Handling Content" on page 187
- "Closing the Connection" on page 191

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"
- "Handling Content"

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:

```
<td>First name:</td>
<td><input type="text" name="FirstName" value=""/></td>
<td>Last name:</td>
<td><input type="text" name="LastName"/ value=""/></td>
...
<td colspan="4" align="center">
   <a href="javascript:on_connect();" >Start chat</a>
      &nbsp;&nbsp;&nbsp;
      <a href="javascript:on_disconnect();" >Stop chat</a>
...
<td colspan="4">
   <textarea cols="80" rows="10" id="transcript"
      name="transcript"></textarea>
</td>...
```

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"
- "Indirect Functions for User Events"
- "Functions to Handle User Events"

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="ChatCommand.aspx.cs"
   Inherits="ChatSample_ChatCommand" validateRequest="false" %>

<!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" href="../../mcr_style.css" type="text/css"/>
<title>MCR Samples 7.6.1 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.
- on_refresh()—calls the on_refresh() function on the page residing in the itf frame.
- parent.main.CommandFrameReady()—boolean flag for making the frame ready to accept a command.
- message_onkeypress()—sets the bCommandFrameReady variable to false and calls the on_user_typing() function on the page residing in the itf frame.

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" />
...
```

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.

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;
```

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:

```
LoadBalancer.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.";
   LoadBalancer.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);
   LoadBalancer.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 != "")
   {
      LoadBalancer.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!";
```

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 107.)

```
LoadBalancer.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)
   {
      LoadBalancer.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;
            LoadBalancer.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.";
            LoadBalancer.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;
         LoadBalancer.LogMessage(LogLevel.Trace, "Can't connect to chat server.
            Reason " + str_itf_message);
      }
      else
      {
         str_itf_message = "Not connected to chat server, unexpected error.";
         LoadBalancer.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() + "')";
   LoadBalancer.LogMessage(LogLevel.Exception, cex.ToString());
```

Handling Content

This section covers these topics:

- "Handling User Requests"
- "Processing Server Events"
- "Masking Data"

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")
{
   LoadBalancer.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
   {
      LoadBalancer.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 transript
         from incoming packet. "+ ex.ToString();
         LoadBalancer.LogMessage
            (LogLevel.Exception, str_itf_message);
   }
}
```

Here is the code to handle a send request:

```
else if (str_cmd == "send")
{
   EventStatus es = null;
   try
   {
      LoadBalancer.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 transript from
            incoming packet. " + ex.ToString();
         LoadBalancer.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 185:

```
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 194
- "Declaring Variables" on page 195
- "Retrieving Form Data" on page 195
- "Connecting to Chat Server" on page 196
- "Creating a Chat Session" on page 197
- "Handling Content" on page 199
- "Closing the Connection" on page 201

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 179. 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"
- "Handling Content"

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:

```
<td>First name:</td>
<td><input type="text" name="FirstName" value=""/></td>
<td>Last name:</td>
<td><input type="text" name="LastName"/ value=""/></td>...

<td colspan="4" align="center">
   <a href="javascript:on_connect();" >Start chat</a>
      &nbsp;&nbsp;&nbsp;
   <a href="javascript:on_disconnect();" >Stop chat</a>...
<td colspan="4">
   <textarea cols="80" rows="10" id="transcript"
      name="transcript"></textarea>
</td>...
```

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"
- "Functions to Handle User Events"

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="ChatCommand.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" href="../../mcr_style.css" type="text/css"/>
<title>MCR Samples 7.6.1 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 107.)

```
   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"
- "Processing Server Events"
- "Masking Data"

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 transript 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 197:

```
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.

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 Java, 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 Java using the E-mail API.
- Submit an e-mail request to E-mail Server Java.
- Report any possible errors in the sample.
- Disconnect from E-mail Server Java.
- Gather submitted information.
- Communicate with the load-balancing service.
- Communicate with E-mail Server Java.
- Generate HTML responses and JavaScript code based on responses from E-mail Server Java.

Functionality Overview

The following sections review the code used in implementing the different e-mail functions:

- “Drawing the Form” on page 203

- “Declaring and Importing Packages” on page 204
- “Retrieving Form Data and Constants” on page 205
- “Processing the Request” on page 205

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 outline 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" href="../../mcr_style.css" type="text/css"/>
<title>MCR Samples 7.6.1 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 border="1">
<tr>
    <td colspan="6">Please enter the interaction information:</td>
</tr>
<tr>
    <td>First name:</td>
    <td>
    <asp:TextBox ID="FirstName" AutoPostBack="false" Text=""
        runat="server" />
    </td>
    <td>Last name:</td>
    <td>
    <asp:TextBox ID="LastName" AutoPostBack="false" Text=""
        runat="server" />
    </td>
    <td>E-mail address:</td>
    <td>
    <asp:TextBox ID="FromAddress" AutoPostBack="false" Text=""
        runat="server" />
    </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.Email;
using Genesyslab.Platform.WebMedia.Protocols.Email.Events;
using Genesyslab.Platform.WebMedia.Protocols.Email.Requests;
using Genesyslab.WebApi.Core.ConfigServer;
using Genesyslab.WebApi.Core;
using Genesyslab.Platform.Configuration.Protocols.ConfServer;
```

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 Java Instance

This section of the `Email.aspx.cs` code creates a load balancer instance and returns an available instance of E-mail Server Java 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 Java upon the submission of each request.

```
try
{
   ServiceInfo si = LoadBalancer.GetServiceInfo
      (CfgAppType.CFGEmailServer, ssc.TenantName);
   svcHost = si.Host;
   svcPort = si.WebApiPort;
   bServiceAvailable = true;
}...
```

If the `try-catch` block is unsuccessful in getting a handle to E-mail Server Java, 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.EmailProtocol email = new
      EmailProtocol(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();

EmailProperties mProperties = new EmailProperties();
mProperties.FirstName = strFirstName;
mProperties.LastName = strLastName;
mProperties.FromAddress = strFromAddress;
if (strMailBox != null && strMailBox != "")
mProperties.Mailbox = strMailBox;   //was ReplyFrom
mProperties.EmailBody = strEmailBody;
mProperties.Subject = strSubject;
```

Preparing the Attachment

The following code block prepares an attachment, if the `abAttachment` variable is not empty. The `Convert.ToBase64String(abAttachment)` function is particularly important—it encapsulates the attachment's binary data in BASE64 format that E-mail Server Java understands.

```
if (abAttachment != null && abAttachment.Length > 0)
{
   mProperties.Attachments = new AttachmentsCollection();
   Genesyslab.Platform.WebMedia.Protocols.Email.Attachment attach =
      new Genesyslab.Platform.WebMedia.Protocols.Email.Attachment();
   attach.Content = Convert.ToBase64String(abAttachment);
   // setting string value to KVlist
   attach.Name = AttachmentField.FileName;
   mProperties.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(reqSubmit, new TimeSpan(0,0,30));
if (im != null && im.Id == EventAck.MessageId)
{
   EventAck eventAck = im as EventAck;
   Interaction_Id.Text = eventAck.RequestId;
   strErrorMessage = "Request " + eventAck.RequestId + " has been successfully
                        submitted to E-mail Server.";
}
else if (im != null && im.Id ==
               Genesyslab.Platform.WebMedia.Protocols.Email.Events.EventError.MessageId)
{
   Genesyslab.Platform.WebMedia.Protocols.Email.Events.EventError eventError =
               im as EventError;
   strErrorMessage = eventError.Description;
}
...
```

Open Media Sample

This sample demonstrates a web form through which users can create and submit interactions involving attached Open 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 Open 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 Open Media Sample:

- "Creating the HTML Header" on page 209
- "Constructing the HTML Body" on page 209
- "Handling User Events" on page 210
- "Declaring and Importing Packages" on page 210
- "Declaring Variables and Importing Constants" on page 211
- "Collecting User Data" on page 211
- "Drawing the Form and Submitting User Data" on page 212
- "Processing the Request" on page 212
- "Spare Event-Handler Prototypes" on page 213
- "Connecting to Interaction Server" on page 214
- "Interaction Event Handlers" on page 215
- "Getting Load Balancer and Interaction Server Instance" on page 212
- "Submitting Requests" on page 215
- "Analyzing Responses" on page 217
- "Closing Connections" on page 217

Files

The ...\ItxSubmit directory contains the Open Media Sample. The sample consists of two files, ItxSubmit.aspx and ItxSubmit.aspx.cs.

Code Explanation

Like the other .NET samples, the Open 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>
<title>Interaction Submit Sample</title>
<meta http-equiv="Content-Type" content="text/html" />
<link rel="stylesheet" href="../../mcr_style.css" type="text/css" />
</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 style="background-image:url(../../fon.gif)" onload="javascript:window_onload();">
<form id="Form2" action="ItxSubmit.aspx" method="post" runat="server">
...
    <td colspan="2" align="center"><h2>Enter information for Interaction Server</h2>
...
<tr>
    <td>Interaction Server host</td>
    <td><asp:TextBox ID="tbServerName" AutoPostBack="false" MaxLength="30" Text=""
   runat="server" /></td>
</tr>
<tr>
    <td>Interaction Server port</td>
    <td><asp:TextBox ID="tbPort" AutoPostBack="false" MaxLength="5" Text=""
   runat="server" /></td>
</tr>
...
<td>Media type:</td>
    <td><asp:TextBox ID="tbMediaType" runat="server"/> <br/>
    <asp:DropDownList ID="selMedia" width="155px" 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:DropDownList>
    </td>
...
<tr>
```

```
    <td colspan="2">
        <input id="btnSubmitInteraction" type="Submit" value="Submit Interaction"
   onserverclick="SubmitInteraction_onClick" runat="server" />
        <input id="btnStopProcessing" type="Submit" value="Stop processing"
   onserverclick="StopProcessing_onClick" runat="server" />
        <input id="btnUpdateInteraction" type="Submit" value="Update Interaction"
   onserverclick="UpdateInteraction_onClick" runat="server" />
    </td>
</tr>

<tr>
    <td colspan="2"><asp:TextBox ID="tbMessages" AutoPostBack="false" Columns="100"
   Rows="10" TextMode="Multiline" runat="server" /></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 Open 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 Open 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);
      iTenandID           = 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 = "Intraction 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 'enpoints' 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 ("Open Media Sample" on page 162), 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)
{
   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 = "MCR_WebAPIServer761";

   try
   {
      itxProtocol.Open();
   }
   catch (Genesyslab.Platform.Commons.Protocols.ProtocolException e1)
   {
      tbMessages.Text = "Can't connect to the Interaction Server.";
      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 occured.\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 succesfully 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 218
- "Constructing the HTML Body" on page 219
- "Declaring and Importing Packages" on page 219
- "Declaring Variables and Importing Constants" on page 220
- "Collecting User Data" on page 220
- "Drawing the Form and Submitting User Data" on page 221
- "Getting Load Balancer and Stat Server Instance" on page 221
- "Spare Event-Handler Prototypes" on page 221
- "Connecting to Stat Server" on page 222
- "Processing the Request" on page 221
- "Event Handlers" on page 223
- "Closing Connections" on page 225

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>
<title>MCR Samples 7.6.1 Stat Server Sample</title>
<meta http-equiv="Content-Type" content="text/html" />
<link rel="stylesheet" href="../../mcr_style.css" type="text/css" />
</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 style="background-image:url(../../fon.gif)
   "onload="javascript:window_onload();">
<form id="Form2" action="StatInfo.aspx" method="post"runat="server">
...
<td colspan="2"
   align="center"><h2>Enter information for StatServer</h2>
...
<td>Please select statistic to display</td>
<td>
   <asp:DropDownList ID="selStatistic" width="250px"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">
         Webform: total distribution time
      </asp:ListItem>
      <asp:ListItem Value="5">
         Webform: queue length
      </asp:ListItem>
      <asp:ListItem Value="6">
         Webform: total distributed
      </asp:ListItem>
   </asp:DropDownList>
</td>...
```

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 dropdown 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 occured.\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 occured. 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 227
- "Constructing the HTML Body" on page 227
- "Declaring and Importing Packages" on page 229
- "Declaring Variables and Importing Constants" on page 229
- "Collecting User Data" on page 230
- "Drawing the Form and Submitting User Data" on page 230
- "Getting Load Balancer and UCS Instance" on page 230
- "Connecting to Interaction Server" on page 231
- "Closing Connections" on page 239

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" href="../../mcr_style.css" type="text/css" />
<title>MCR Samples 7.6.1 UCS</title>
</head>
```

Constructing the HTML Body

Next, the code includes the HTML code to create the input fields for fist name, last name, and e-mail address; the Search button; and to display the query results:

```
<body onload="javascript:window_onload();" style="background-image:url(../../fon.gif)">
<form id="ucs_form" method="post" action="UCS.aspx" runat="server">
  <table border="1" width="30%">
    <tr>
      <td colspan="2" align="center"><h2>Enter information for UCS</h2></td>
    </tr>
    <tr>
      <td>UCS host</td>
      <td><asp:TextBox ID="tbServerName" AutoPostBack="false" MaxLength="30"
        Text=""  runat="server" /></td>
    </tr>
    <tr>
      <td>UCS port</td>
      <td><asp:TextBox ID="tbPort" AutoPostBack="false" MaxLength="5"
        Text="" runat="server" /></td>
    </tr>
    <tr>
      <td>First name:</td>
      <td><asp:TextBox ID="tbFirstName" AutoPostBack="false"
        MaxLength="30" Text="begemot" runat="server" /></td>
    </tr>
    <tr>
      <td>Last name:</td>
      <td><asp:TextBox ID="tbLastName" AutoPostBack="false" MaxLength="30"
        Text="" runat="server" /></td>
    </tr>
    <tr>
      <td>E-mail:</td>
      <td><asp:TextBox ID="tbEMail" AutoPostBack="false"
        MaxLength="30" Text="begemot@aspirin" runat="server" /></td>
    </tr>
    <tr>
      <td colspan="2">
      <input id="btnSearch" type="Submit" value="Search"
```

```
            onclick="JavaScript:return SearchButton_onClick();
            "onserverclick="Search_onClick" runat="server" />
          </td>
        </tr>
      </table>
      <br/>
      <br/>
      <table border="1" width="30%">
      <tr>
        <td colspan="2" align="center">
        Color diagram
        </td>
      </tr>
      <tr>
        <td align="center" style="background-color:<%=threadBackgroundColorNew%>">
          &nbsp;&nbsp;&nbsp;&nbsp;
        </td>
        <td align="left">
          - Thread with new messages
        </td>
      </tr>
      <tr>
        <td align="center" style="background-color:<%=threadBackgroundColorOld%>">
          &nbsp;&nbsp;&nbsp;&nbsp;
        </td>
        <td align="left">
          - Thread with no new messages
        </td>
      </tr>
      <tr>
        <td align="center" style="background-color:<%=emailFromCustomerColor%>">
          &nbsp;&nbsp;&nbsp;&nbsp;
        </td>
        <td align="left">
          - Message from customer
        </td>
      </tr>
      <tr>
        <td align="center" style="background-color:<%=emailFromAgentColor%>">
          &nbsp;&nbsp;&nbsp;&nbsp;
        </td>
        <td align="left">
          - Message from call center
        </td>
      </tr>
    </table>

      <h2>
        <asp:Label ID="ResultData" runat="server"></asp:Label>
      </h2>...
```

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.Criterias = new SearchCriteriaCollection();
      csc.Criterias.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 ("\r\n <table border=\"0\"
   width=\"100%\" id=\"treadsTable\" cellpadding=\"5\" style=\"border-collapse:
   collapse\">\r\n");
   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();
```

Chapter

11 Multimedia Compound Sample

This chapter explains the Multimedia Compound Sample. The current release provides this sample for Java only. The chapter contains the following sections:

Overview

The Compound Sample is a simple web application that demonstrates how to use the Multimedia web media features such as chat, e-mail, web collaboration, voice callback, and FAQ in a secure environment.

Note: The Compound Sample presents the Web API media features in one cohesive application. That's why it was created. Most of the media code is taken from the web samples described in Chapter 9. You should read that chapter before reading this one and make sure you understand how each of the media features works. Pay particular attention to the code explanations.

Files and Directory Structure

`/CompoundSample761`, located under the `<tomcat_home>/webapps` directory, is the root directory of the Compound Sample. It contains several subdirectories, which are listed in Table 9 on page 242.

Table 9: Names and Purposes of the Compound Sample Directories

Directory	Purpose
AccountInfo	Contains a JSP file that asks a user to fill out personal information
Chat	Contains the web Chat Sample. “Chat Sample” on page 107 explains this sample in detail.
Callback	Contains the web Callback Sample. “Callback Sample” on page 101 explains this sample in detail.
Email	Contains the web E-mail Sample. “E-Mail Sample” on page 120 explains this sample in detail.
Help	Contains help files that assist a user with a particular page. Most of the sample pages have a `Help` icon that launches a help file.
Login	Contains files for logging in and logging out of the Compound Samples and files used to validate user names and passwords.
Images	This directory holds 25 graphics files for menu tabs, Server Availability icons, phones, and so on. If you want to add graphics, you should place and reference them here.
MainWindow	Contains the main greeting or entry point to the Compound Sample
META_INF	Contains the `manifest.mf` file. A web server autogenerates this file.
WEB_INF	Contains a mandatory configuration file for all web applications: `web.xml`. This file contains only the default content of all web applications. It does not contain any customized instructions for the Compound Sample.

Sample Demonstration Categories

The Compound Sample has two demonstration categories:

- `Authentication`
- `Media Availability`

Authentication

The Compound Sample demonstrates how to:

- Create a simple login function with predefined `username` and `password` values.
- Emulate access to a database and provide a customer's personal information from the DB.
- Create common reusable code that is included in each JSP and prevent its unauthorized use.

Warning! The authentication scheme used in the Compound Sample is for illustration only and therefore very basic. If you are building a data-sensitive application, you must use a more secure authentication scheme.

Media Availability

The Compound Sample demonstrates how to:

- Create a dynamic menu with currently available media.
- Show only the media available based on a status report from Genesys Local Control Agent.

Common Files

You will find nine common files under the Compound Sample root directory. This section groups these files in categories and explains each file in detail.

- `ApplicationConstants.jsp`
- `blank.jsp`
- `CheckCorrectLogin.jsp`
- `CommonScripts.jsp`
- `JSConstants.jsp`
- `calendar.js`
- `CommLib.js`
- `icc_style.css`
- `index.html`

Placeholder

The blank.jsp file is used as a placeholder in the bottom-right frame of the Compound Sample. When the frame is initialized, blank.jsp is either replaced by PersonalInfo.jsp or by the page specified in the RedirectToPage request parameter. blank.jsp is also used for the hidden frame that is used by the Compound Sample. (See “Compound Sample Structure” on page 246.)

Calendar Utility

The calendar.js file contains JavaScript utility functions for displaying calendars and formatting dates.

Common Library

The CommLib.jsp file acts as the common API library, and the CommonScripts.jsp file acts as the common JavaScript code for all the samples.

The CommLib.jsp file contains functions that:

- Identify a user’s web browser.
- Get handles to HTML frames or control objects.
- Perform basic string manipulation and encoding.
- Retrieve submitted form parameters.
- Return the current time.

The CommonScripts.jsp file contains java methods that:

- Mask unacceptable characters.
- Generate HTML buttons.
- Check for media availability.

For example, the IsMediaAvailable method allows you to set the availability of a media type:

```
public boolean IsMediaAvailable (int iType)
{
   boolean bReturn = false;
   SvcDispatcher svcDispatcher = null;
   try
   {
      svcDispatcher = new SvcDispatcher();
      if(svcDispatcher != null && svcDispatcher.getErrorCode() == 0)
      {
         if (svcDispatcher.inqSrvcByType(iType, strTenant))
         {
            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);
```

In the following switch statement, the chat media type and been made available Monday through Friday from 10AM until 6PM:

```
switch (iType)
{
   case CfgAppType.CFGChatServer:
      if (iCurDayOfWeek != Calendar.SUNDAY & iCurDayOfWeek
            != Calendar.SATURDAY)
         if (iCurHour >= 10 && iCurHour <= 18)
            bReturn = true;
      break;
   case CfgAppType.CFGEmailServer:
      bReturn = true;
      break;
   case CfgAppType.CFGUniversalCallbackServer:
      bReturn = true;
      break;
   default:
      break;...
```

Constants

The Compound Sample has two constant files, ApplicationConstants.jsp and JSConstant.jsp. Table 10 shows only the configuration constants in the ApplicationConstants.jsp file. The Section Name column lists the database sections under the Options tab of the Multimedia Web API Server Application object in Configuration Server. The second column lists option names. The third column lists the corresponding Java variable name in the ApplicationConstants.jsp file. The last column describes the option named in column two.

Table 10: Configuration Settings in ApplicationConstants.jsp File

Section Name	Field Name	Java Constant Name	Description
chat	chat-queue	strChatQueue	Chat queue
chat	chat-stat-interval	strChatStatInterval	Interval at which Stat Server calculates statistics on chat requests
e-mail	email-queue	strEmailQueue	E-mail queue
e-mail	email-stat-interval	strEmailStatInterval	Interval at which Stat Server calculates statistics on e-mail requests
miscellaneous	applets-code-base	strCodeBase	Code base for all used applets

Table 10: Configuration Settings in ApplicationConstants.jsp File (Continued)

Section Name	Field Name	Java Constant Name	Description
miscellaneous	stat-refresh-interval	strStatRefreshInterval	Interval for a statistics refresh on statistics pages
miscellaneous	tenant	strTenant	Tenant in Configuration Server

Authentication

The `CheckCorrectLogin.jsp` file demonstrates the `authentication` section of the sample. See "Authentication" on page 250 for a detailed discussion of the security features in the Compound Sample.

Presentation

Greetings Page

`The index.html` file is the main or greeting page to access the Compound Sample. The file presents an HTML link to the login page. Figure 26 on page 248 shows how the file looks after launching.

Graphics

The graphics are all under the `Images` directory. See the Images entry in Table 9 on page 242 for more information.

Style Sheet

The cascading style sheet (CSS) `icc_style.css` has instructions for adding the bold font to header tags, adding tables, and other layout code. This document does not discuss CSS technologies. You should be able to easily find CSS tutorials on the Web.

Compound Sample Structure

You can access the Compound Sample either through the main `index.html` page (see Figure 26 on page 248) or `MainFrame.jsp` (see Figure 28 on page 249). In either case, you will be forwarded to the login page (see Figure 27 on page 248) if your credentials have not yet been authenticated.

The center stage of the Compound Sample is the `MainFrame.jsp` file. This file is under the `/MainWindow` directory and it contains JavaScript functions that process keys used in the sample. The file also contains a frameset that holds the `LeftNavFrame.jsp, Command.jsp,` and `PersonalInfo.jsp` files, as well as a hidden frame containing `blank.jsp.` Note that on loading, the frameset contains blank.jsp in the BottomMainFrame. This will be replaced with PersonalInfo.jsp when the page is initialized. Here is the HTML frameset code:

```
<frameset cols="170,*,0" frameborder="NO" border="0" framespacing="0"
    rows="*" onload="javascript:init();">
  <frame name="LeftNavFrame" scrolling="NO" noresize
    src="../MainWindow/LeftNavFrame.jsp">
  <frameset rows="75,*, 0" frameborder="yes" border="1"
    framespacing="0" cols="*">
    <frame name="UpperFrame" noresize src="../MainWindow/Command.jsp">
    <frame name="BottomMainFrame" noresize src="../blank.jsp"">
  <frame name="SystemHiddenFrame" scrolling="NO" noresize src="../blank.jsp">
  </frameset>
</frameset>
```

The left frame, LeftNavFrame, contains the site navigator file LeftNavFrame.jsp. The main frame on the bottom-right of the page contains input boxes. The top-right frame holds the Command.jsp file that contains menu tabs. You will see different tabs depending on what media are available. The main frame changes to load different pages. Both the navigator and top-right frame command tabs can affect the content displayed in the center. Figure 25 shows the layout of the sample.

Note: The frame layout shown in Figure 25 does not reflect the proportions you will see in your browser. In particular, the hidden frame on the right will not show up on your screen.

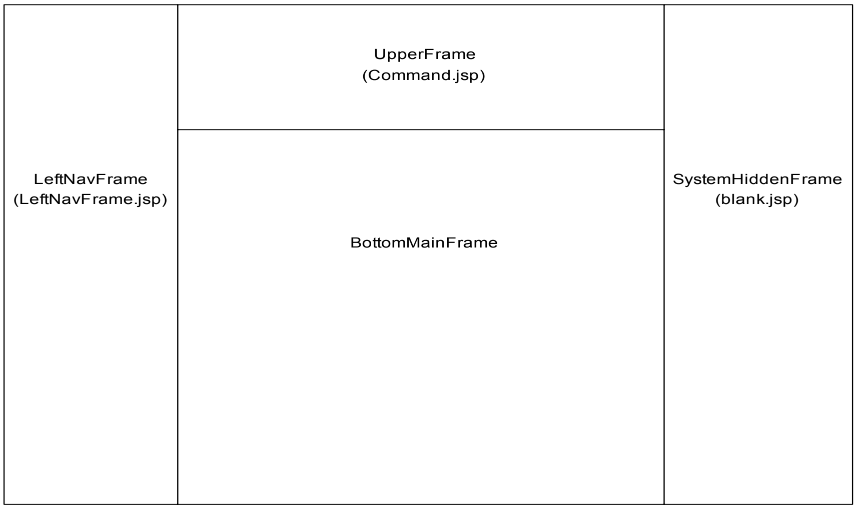
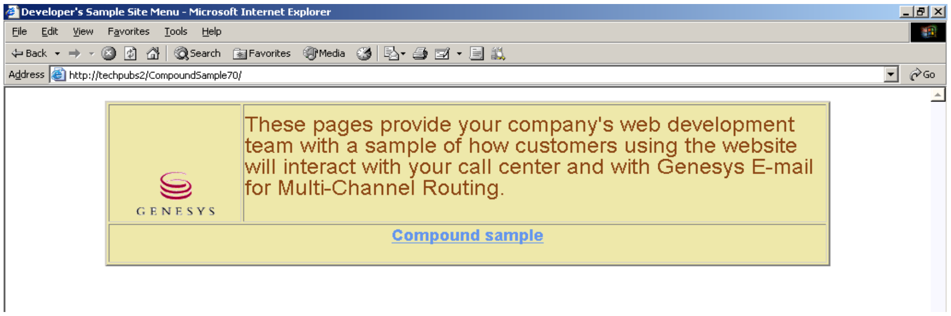


Figure 25: Compound Sample Layout

Running the Sample

The primary entry point to the Compound Sample is through the `index.html` file. Figure 26 shows how the file looks after you launch it.


Developer's Sample Site Menu - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address
Go
GENESYS

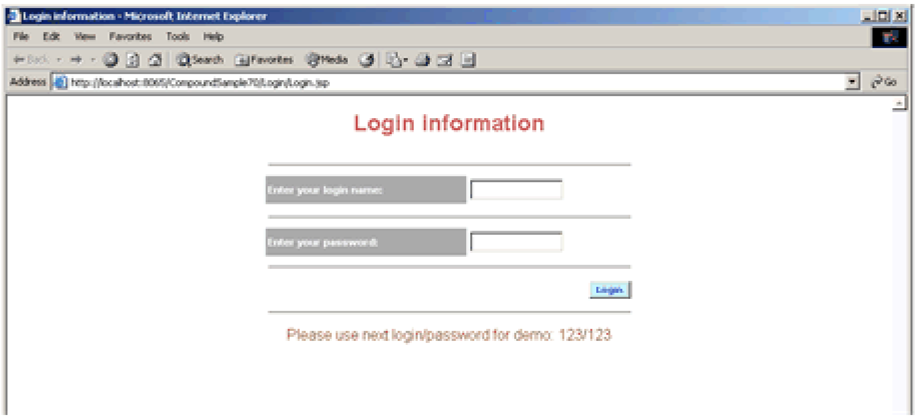
http://techpubs2/CompoundSample70/

These pages provide your company's web development team with a sample of how customers using the website will interact with your call center and with Genesys E-mail for Multi-Channel Routing.

Compound sample

Figure 26: Main Greeting Page of the Compound Sample

You must click the `Compound Sample` link to launch the application. After clicking the link, you are taken to a login page (see “Authentication” on page 243 for a code explanation), as shown in Figure 27. You must log in to continue.


Login information - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address
Go
Login

http://localhost:8065/CompoundSample70/Login/Login.jsp

Login information

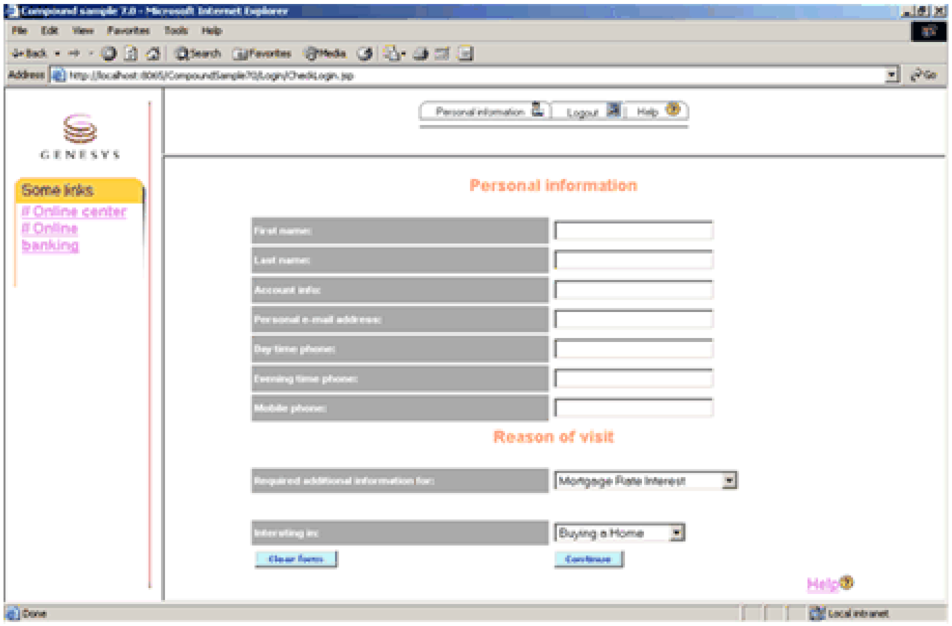
Enter your login name:

Enter your password:

Please use next login/password for demo: 123/123

Figure 27: Login Page for the Compound Sample

Eventually, the user arrives at `MainFrame.jsp`, the main page of the Compound Sample application (see Figure 28 on page 249). If you attempt to access this file directly without authentication, you are forwarded to the login page shown in Figure 27.


Compound sample 7.0 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8065/CompoundSample70/Login/CheckLogin.jsp
Go
Personal information
Logout
Help
GENESYS
Some links
// Online center
// Online banking
Clear form
Continue
Help
Done
Local intranet

Personal information

First name:	
Last name:	
Account info:	
Personal e-mail address:	
Day time phone:	
Evening time phone:	
Mobile phone:	

Reason of visit

Required additional information for:	Mortgage Rate Interest
Interesting in:	Buying a Home

Figure 28: Main Page for the Compound Sample

Fill out the form on the bottom-right frame and click `Continue`. If you filled out the form correctly, the `Continue` button opens a new page with a listing of the media features available to you at that moment. You should also see extra tabs on the top-right pane for each of the available media. You can click these tabs to access a particular page directly.

Note: The media servers are not always available. The sample includes a demonstration of that point. Issues like communication traffic, office hours, agent availability, client and server error, or network problems can all contribute to the lack of service.

Customizing the Compound Sample

You can modify or add HTML links to point to other new pages, replace the background color or design, company logo, and add other branding items on the main page of the Compound Sample. You can also add or modify HTML controls on the pages.

Note: Keep in mind that the purpose of the Compound Sample is to demonstrate the Multimedia media functions and features in one cohesive application and to be a tool to help you build your *own* application. You should not modify the sample for production use because most of its components are sample code themselves (from Chapter 9).

Code Explanation

This section reviews the code for these features of the Compound Sample:

- "Authentication" on page 250
- "Chat" on page 253
- "E-Mail" on page 256

Authentication

Two files handle authentication in the Compound Sample. `Login.jsp` presents a GUI to you to enter your credentials. After the data entry, the page forwards you to the `CheckLogin.jsp` file, which contains the actual authentication code. The following subsections outline the key lines of code in each file.

Warning! The authentication scheme used in the Compound Sample is for illustration only and therefore very basic. If you are building a data-sensitive application, you must use a more secure authentication scheme.

Logging In

The following code snippets are from the `Login.jsp` file. The line below forwards the data entered on the login page to the `CheckLogin.jsp` page:

```
<FORM Name="LoginForm" AutoComplete="off" METHOD="POST"
                            ACTION="../Login/CheckLogin.jsp">
```

This is the HTML code for the input boxes for user name and password:

```
<INPUT TYPE="text" NAME="userid" SIZE="13" MAXLENGTH="11">
<INPUT TYPE="password" NAME="password" SIZE="13" MAXLENGTH="8">
```

The Java code calls the `GenerateButton()` method from the `CommonScripts.jsp` file and passes in the `HTTPServletRequest` and the output stream (in this case, the `JSPWriter`) arguments:

```
<%GenerateButton ("Login", "Login_onClick();", request, out);%>
```

The `GenerateButton()` method shown below checks the browser type, such as Internet Explorer or Netscape, that the user is using and generates HTML code applicable to the browser. The `request.getHeader()` call returns the browser type. The try-catch block then decides which HTML code to write to the output stream:

```
public void GenerateButton (String strTitle, String strJavaScript,
                        HttpServletRequest request, JspWriter out){

   String strBrowser = request.getHeader ("User-Agent");
   String strOut = "";
   try{
      if (strBrowser.indexOf ("MSIE") != -1){
         strOut += "<input type=\"button\" class=\"coolButton\"value=\"";
         strOut += strTitle + "\" onClick=\"javascript:"+strJavaScript + "\">";
      }else {
         strOut += "<a href=\"javascript:" + strJavaScript + "\">" +strTitle + "</a>";
      }
      out.write (strOut);
   }catch (Exception e){}
}
```

If the user makes a login error, the code in the `Login_Onclick()` method prompts the user to reenter the credentials. Otherwise, it submits the form.

```
function Login_onClick()
{
   if (document.forms[0].userid.value == "")
   {
      alert ("Please enter login information.");
      return;
   }
   if (document.forms[0].password.value == "")
   {
      alert ("Please enter password.");
      return;
   }
   document.forms[0].submit();
}
```

Getting Authenticated

The `CheckLogin.jsp` code implements a simplified verification of user credentials:

```
String strUserID= i18nsupport.GetSubmitParametr(request, "userid");
String strPassword= i18nsupport.GetSubmitParametr(request,
   "password");
if (strUserID != null && strUserID.equals("123")){
   session.putValue("login", "true");
%>
```

If the user enters the data required, the JSP forwards the user to the `MainFrame.jsp` page. Otherwise, it returns the user to the login page:

```
<jsp:forward page="../MainWindow/MainFrame.jsp"/>
<% } %>
<jsp:forward page="../Login/
Login.jsp?Error=Incorrect+login.+Please+try+again."/>
```

Web Media Features in the Compound Sample

The Compound Sample shares much of the media-related code from the simple web samples described in Chapter 9. However, the Simple Samples are stand alone demonstrations, which is reflected in the code. On the other hand, the Compound Sample is an application, which required some modification of the shared code to make it function as a whole.

Callback

The following files are used to demonstrate the callback features of the Compound Sample:

- CallbackOptions.jsp—checks for callback availability and presents the callback JSP only if the callback service is available.
- HtmlCallback.jsp—provides functionality similar to the Callback JSP used in the Simple Sample.
- Calendar.jsp—provides a JavaScript calendar window.

Code Explanation

You must read the Simple Callback Sample for a complete explanation of the basic code, as this section focuses only on how the code is modified for the Compound Sample.

Minor Differences

Some minor differences occur between the Simple Sample and Compound Sample versions, such as variable names and parameters.

The Compound Sample also imports other JSP files—CommonScripts.jsp and CheckCorrectLogin.jsp:

```
<%@ include file="../CommonScripts.jsp" %>
<%@ include file="../CheckCorrectLogin.jsp" %>
```

The second statement checks for proper authentication and forces the user back to the login page if he or she tries to access the callback page directly.

Main Differences

The Compound Sample uses files that are very similar to the ones used by the Simple Sample. The overall code and logic are basically the same, with a few differences:

- Compound Sample's version of the `HTMLCallback.jsp` file contains code for authentication (it uses the `CheckCorrectLogin.jsp` file). It also uses the style sheet for the Compound Sample.
- Several JavaScript functions are different, as noted in the next section.

Handling User Events

Several of the JavaScript functions used in the Compound Sample are different from those used in the Simple Callback Sample.

The following functions are essentially the same in both samples:

- `window_onload()`
- `window_onunload()`
- `on_view_list()`
- `on_request()`

These functions are different:

- `selPhoneNum_onChange()`—sets the media type to `voip` if the selected phone number is equal to the client IP address. Otherwise, it sets the media type to `voice.`
- `GetStartDate()`—gets the requested callback start date.
- `GetEndDate()`—gets the requested callback end date.

Chat

One of the media features in the Compound Sample is chat. These files are used to demonstrate it:

- `ChatOptions.jsp`—checks for chat media availability and presents a chat window only if chat is available.
- `ChatTranscript.jsp`—supports popular Internet expressions such as smiling or frowning facial expressions and hyper links in an e-mail or chat communication.
- `HtmlChatCommand.jsp`—contains Java and JavaScript code that controls the Media Options tab in the top-right frame.
- `HtmlChatFrameSet.jsp`—a frameset that holds the `HtmlChatCommand.jsp` and `HtmlChatPanel.jsp` files.
- `HtmlChatPanel.jsp`—contains the chat panel where users can enter and see the chat transcript.

You must read the simple Chat Sample for a complete explanation of the basic code because this section focuses only on how the code is modified for the Compound Sample.

Minor Differences

Some minor differences occur between the Simple Sample and Compound Sample versions, such as variable names and extra parameters for queue-related information.

The Compound Sample also imports other JSP files—`CommonScripts.jsp` and `CheckCorrectLogin.jsp`:

```
<%@ include file="../CommonScripts.jsp" %>
<%@ include file="../CheckCorrectLogin.jsp" %>
```

The second statement checks for proper authentication and forces the user back to the login page if he or she tries to access the chat page directly.

Main Differences

The Compound Sample uses chat files that are very similar to the ones used by the Simple Sample. The overall code and logic are basically the same, with a few differences:

- Compound Sample chat includes an extra file—`ChatOptions.jsp`. The `Command.jsp` file uses it to check for different chat media availability, such as chat or a combination of chat and another media.
- Compound Sample's version of the `HTMLChatPanel.jsp` file contains code for authentication (it uses the `CheckCorrectLogin.jsp` file). It also uses the style sheet for the Compound Sample.
- Compound Sample's version of the `ChatStatInfo.jsp` file contains code that references the constants from the `ApplicationConstants.jsp` file:

  ```
  <%include file="../ApplicationConstants.jsp" %>
  ```

  It also adds an extra line to call the `GetDataFromConfigServer()` method in `ApplicationConstants.jsp` to retrieve option values from the Compound Sample `Application` object in Configuration Server:

  ```
  <%GetDataFromConfigServer ();%>
  ```

Chat Box

The main difference in the chat panel is the implementation of emotion icons and hyper links. This implementation is contained within the four methods of the ChatTranscript.jsp:

- AddMessage()—adds messages from server to the chat transcript panel.
- ParseSmilesAndLinks()—parses message to be displayed. Shows links as clickable hyper-links and smiles as emoticons.
- ProcessOneWord()—checks each word in the message to determine if it is a an emoticon or a hyper-link.
- HideHTML()—used to prevent hacking by hiding all potentially dangerous content, such as these symbols: < ” > &. This makes the incoming message harmless to the client.

An array of GIF files found in the ChatTranscript.jsp 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 hyper link it must begin with one of the following formats:

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

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

Chat Command

There is only one main modification of the HtmlChatCommand.jsp—the on_connect() function. The Compound Sample stores the information the user entered in the top-right frame of the sample. The following code snippet shows the modified section:

```
function on_connect()
{
   clearTimeout(timerID);
   document.forms[0].cmd.value             =   "connect";
   document.forms[0].first_name.value      =   top.GetValue("FirstName");
   document.forms[0].last_name.value       =   top.GetValue("LastName");
   document.forms[0].email_address.value   =   top.GetValue("FromAddress");
   document.forms[0].queue_key.value       =   top.GetValue("<%=fldnInterest%>");
   document.forms[0].subject.value         =   top.GetValue("Subject");
   var RightNow=new Date();
   document.forms[0].timeZoneOffset.value  =   RightNow.getTimezoneOffset();
   document.forms[0].submit();
}
```

E-Mail

The code for the e-mail feature in the Compound Sample is basically the same as that for the simple E-mail Sample; however, there a few differences. For example, in the simple E-mail Sample you are required to enter all of the information on the form, while the Compound Sample stores this information for the user. The Compound Sample demonstrates the E-mail History functionality and the ability to print this history. It also enables us to link the reply to an e-mail that has its parent message stored in the Universal Contact Server Database.

One of the media features in the Compound Sample is e-mail. These files are used to demonstrate it:

- `Email.jsp`—contains Java code snippets that use the e-mail portion of the Web API to send e-mails.
- `EmailHistory.jsp`—gets customer interactions history from Universal Contact Server.
- `EmailHistoryFrameset.jsp`—the frameset for the e-mail history functionality.
- `EmailOptions.jsp`—shows all available e-mail options, like web form submission or e-mail history, from Universal Contact Server.
- `PrintHistory.jsp`—prints selected thread or single e-mail. Also helps to update attach data of the interaction.

The majority of the code pertaining to the e-mail history and the print history functionality is contained in the `EmailHistory.jsp` and the `PrintHistory.jsp`.

The PrintHistory.jsp File Explained

The first line in the `PrintHistory.jsp` file sets the page content type:

```
<%@ page contentType = "text/html; charset="windows-1252" %>
```

The second line is a call to the `response.setContentType()` method. This sets the content type or character encoding to use in the response to the client:

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

Then the sample code loads the following libraries into memory:

```
<%@ page import="Genesys.webapi.system.loadbalancing.*"%>
<%@ page import="Genesys.webapi.media.irs.direct.*"%>
<%@ page import="Genesys.webapi.media.irs.protocol.*"%>
<%@ page import="Genesys.webapi.media.common.*"%>
<%@ page import="Genesys.webapi.utils.i18n.*"%>
<%@ page import="Genesys.CfgLib.*"%>
<%@ page import="Genesys.webapi.media.ucs.direct.*"%>
<%@ page import="java.io.*"%>
<%@ page import="java.util.*"%>
<%@ include file="../CommonScripts.jsp" %>
<%@ include file="../CheckCorrectLogin.jsp" %>
```

The getValueFromMap method is used to retrieve any value of the key by the specified name:

```
<%!
public String getValueFromMap (Map map, String strName)
{
//Useful attributes: Text, ThreadId, Subject, FromPersonal, Mailbox,
//SentDate, StartDate, EndDate

  String strBody = null;
  _ucs_attribute atrBody = (_ucs_attribute) map.get(strName);
  if (atrBody != null)
  {
    Object oBody = atrBody.getValue();
    if (oBody != null)
      strBody = oBody.toString();
  }
  if (strBody == null)
    strBody = "";
  return strBody;
}
%>
```

Then the code creates a relationship to a style sheet:

```
<link rel="stylesheet" href="/CompoundSample761/icc_style.css"type="text/css">
```

The code uses two external JavaScript files:

```
<SCRIPT LANGUAGE=javascript SRC="/CompoundSample761/CommLib.js"><SCRIPT>
<SCRIPT LANGUAGE=javascript SRC="/CompoundSample761/JSConstants.jsp"></SCRIPT>
```

Variables are declared:

```
<%
    String action = i18nsupport.GetSubmitParametr(request, "action");
    String thread_id = i18nsupport.GetSubmitParametr(request, "thread_id");
    String email_id = i18nsupport.GetSubmitParametr(request, "email_id");
    String contact_id = i18nsupport.GetSubmitParametr(request, "contact_id");
%>
```

An instance of the load balancer is created and a Universal Contact Server is selected for each request. The remaining code allows you to print all of the e-mail history or one specific e-mail. The file also contains the logic needed to create the color coded table that the e-mail history is displayed in.

```
<%
   if(action != null)
   {
      String svcHost = "";
      int svcPort = -1;
      svcDispatcher = new SvcDispatcher();
      _ucs_direct search = null;

      // We select UCS server for each request(!)
      if( svcDispatcher != null && svcDispatcher.getErrorCode() == 0 &&
        svcDispatcher.inqSrvcByType(CfgAppType.CFGContactServer, strTenant) )
      {
        svcHost = new String(svcDispatcher.getSrvcHost().toLowerCase());
        svcPort = svcDispatcher.getSrvcUCSApiPort();
        search = new _ucs_direct();
        search.connect(svcHost, svcPort);
      }

      List emins = null;
      _ucs_interaction one_mail = null;
      if (search != null )
      {
        if (action.equals("mark_as_read"))
        {
          one_mail = search.get_one_interaction(email_id);
          _ucs_parameter_map userdata = one_mail.getInteractionAttributes();

          if (userdata == null)
            userdata = new _ucs_parameter_map ();
          userdata.put (strWebSafeEmailStatus, "Read");

          search.update_interaction (email_id,
           svcDispatcher.getTenantId(strTenant).intValue(), userdata);

          out.println("<H2 align=\"center\">Email has been marked as read.
           Please refresh emails history frame</H2>");
         out.println ("<A HREF=\"JavaScript:window.opener.Submit_onClick(0);close();\">
           Close this window and refresh history</A>");
        }
        else if (action.equals("print_thread"))
        {
          emins = search.get_interactions_thread(thread_id);
          out.println("<H2 align=\"center\">Email thread history</H2>");
        }
        else if (action.equals("print_email"))
        {
           one_mail = search.get_one_interaction(email_id);
           Vector aVec = new Vector();
           aVec.add (one_mail);
           emins = (List) aVec;
           out.println("<H2 align=\"center\">Email info</H2>");
        }
        else
        {
          out.println("<H2 align=\"center\">
           Unsupported action command performed.</H2>");
        }
      }...
```

The EmailHistory.jsp File Explained

Most of the code in this file has been explained in detail within the explanation of the `PrintHistory.jsp`. The majority of the code in this file creates the color coded table that the e-mail history is displayed in, but there are a few new functions within this file:

- `Submit_onClick()`—submits an e-mail.
- `Reset_onClick()`—clears the fields of the e-mail form.
- `ClickOnRow()`—expands and collapses a row in the e-mail history table.
- `MarkAsRead()`—marks an e-mail as read.
- `PrintThread()`—prints the e-mail history.
- `PrintOneEmail()`—prints a single e-mail.

Linking E-mail Replies with Parent Messages

Code changes in the `Email.jsp` and the `EmailHistory.jsp` files that are not included in the Simple Sample enables linking of the customer's reply to the exact parent message in the Universal Contact Server Database.

Note: To attach a reply to the exact parent e-mail, you must add the parameter `ParentId`. The `Id` of the original e-mail message must be included in a parameter passed by the `submit` method.

See the following code in the `Email.jsp` file.

First, create a `parentId String` variable and set to the return value of the `GetSumitParametr` method:

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

Next, create a hidden variable:

```
<tr>
    <td colspan=6>
        <textarea cols="100" rows="10"
   NAME="<%=mask_html(fldnEmailBody)%>"><%=mask_html_for_textarea(body)%></textarea>
        <input type="hidden" name="ParentId" value="">
    </td>
</tr>
```

If the there is a parent message related to this reply, add the message to the `userdata` information:

```
if (parentId != null && parentId.equals("") == false)
   userdata.addElement(new _kvitem("ParentId", parentId));
```

Use the `setReplyAttributes` function to add the `parentId` value to the `ParentId` attribute of the form:

```
function setReplyAttributes (parentId)
{
   document.forms[0].ParentId.value = parentId;
}
```

Note the changes made to the `EmailHistory.jsp` file, shown below, for this functionality.

Obtain the Interaction ID using the `getValueFromMap` method and put the value into the `String`, `emailID`:

```
String emailID = getValueFromMap (map, "Id");
```

Call the `GenerateJavaScript` and pass in the `emailID` as a parameter:

```
GenerateJavaScript (out, strSubject, emailBody, i, emailID);
```

See the following modified `GenerateJavaScript` function that includes a `String strParentId` parameter:

```
public void GenerateJavaScript (JspWriter out, String strSubject, String strBody, int
   iNumber, String strParentId)
{
  try
  {
    out.println ("<script language=\"JavaScript\">");
    out.println ("function ReplyToEmail" + iNumber + "()");
    out.println ("{");
    out.println ("  parent.WebFormFrame.document.forms[0]."+ fldnSubject + ".value =\""
        + MaskSymbols("Re: "+strSubject) + "\";");
    out.println ("  parent.WebFormFrame.document.forms[0]."+ fldnEmailBody + ".value
        =\"" + MaskSymbolsForReply(strBody) + "\";");
```

The next line in the function adds the `strParentId` to the reply:

```
    out.println ("  parent.WebFormFrame.setReplyAttributes (\"" + strParentId + "\");");
    out.println ("}");
    out.println ("</script>");
  }
  catch (IOException e)
  {
  }
}
```

Chapter

12 Multimedia Test Tools

This chapter describes the Multimedia test tools, in three sections:

- Overview, page 263
- Using the Tools, page 265
- Troubleshooting Guide, page 269

Overview

This chapter shows how to access the test tools provided with the Multimedia web sample installation. The installation wizard prompts you to install these tools.

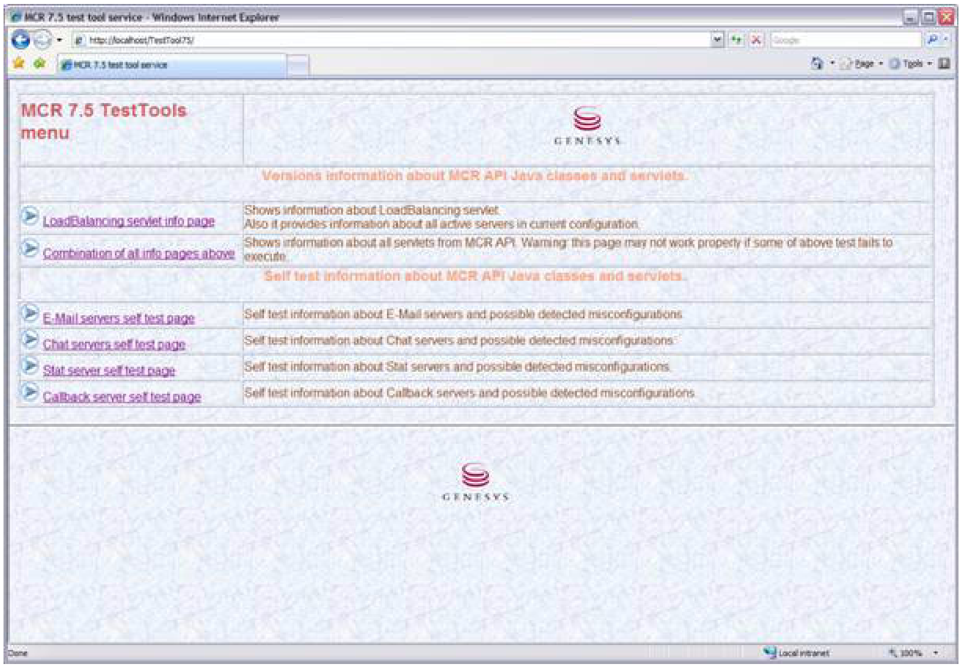
This chapter does not discuss how the customized server-page files are constructed, nor the code involved in creating them.

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>/
TestTool761/index.html
```

where `<web_server_hostname>:<web_server_port>` is the host and port information for the machine on which you installed the Multimedia test tools. Figure 29 shows the menu page. Notice that the top frame contains a scrollable list of test tools.


MCR 7.5 test tool service - Windows Internet Explorer
http://localhost/TestTool75/
MCR 7.5 test tool service
Page
Tools
GENESYS
Done
Local intranet
100%

MCR 7.5 TestTools menu

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.
Combination of all info pages above	Shows information about all servlets from MCR API. Warning: this page may not work properly if some of above test fails to execute

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
Callback server self test page	Self test information about Callback servers and possible detected misconfigurations

Figure 29: Test Tools Menu Loaded Successfully

Warning! Genesys recommends that you restrict public access to your host machine's entire `.../TestTool761` or `...\TestTool761` 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 7.6 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>/WebAPIServer761/
TestTool761/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 171.)
2. Then click `Loadbalancing informational page` link on that page.

Either way, Figure 31 on page 266 shows the resulting test page.

Using the Tools

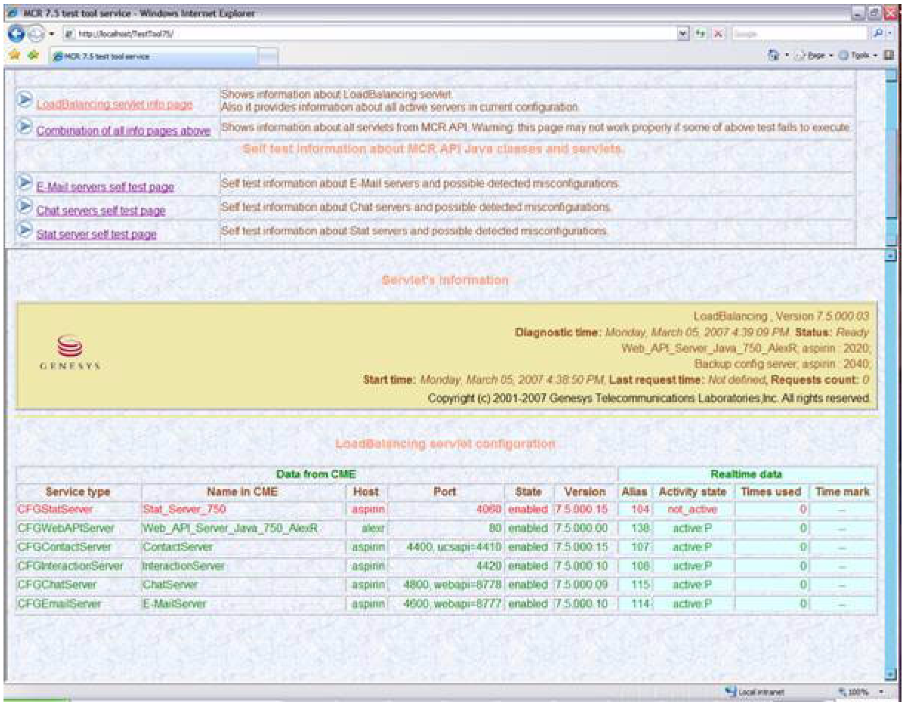
Use the tools to verify the configuration of your media servers and to see which media servers are available. The test tools verify the configurations for these components:

- Load-Balancing Servlet
- E-mail Server Java
- Stat Server
- Voice Callback Server (see the note on page 265)
- Chat Server

Note: This chapter does not discuss the configuration tests for Voice Callback Server and the combination of all Multimedia servlets.

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 30 shows a successful result from clicking on the first link.


MCR 7.5 test tool service - Windows Internet Explorer
http://localhost/TestTool75/
MCR 7.5 test tool service
Page
Tools
Local intranet
100%

LoadBalancing servlet info page	Shows information about LoadBalancing servlet. Also it provides information about all active servers in current configuration.
Combination of all info pages above	Shows information about all servlets from MCR API. Warning: this page may not work properly if some of above test fails to execute.

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.

Servlet's information

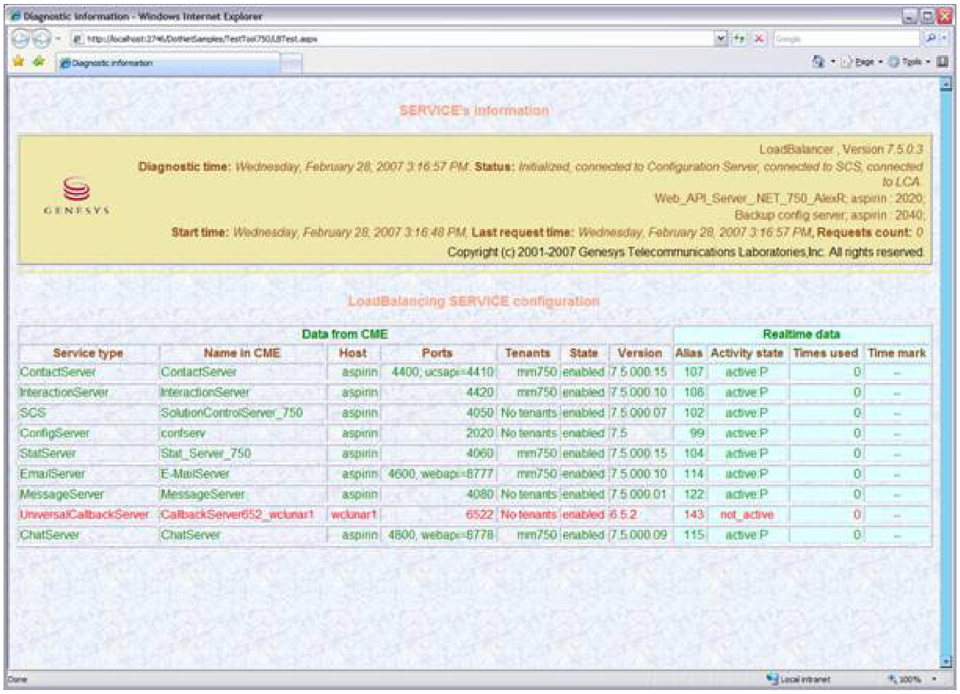
LoadBalancing , Version 7.5.000.03
Diagnostic time: *Monday, March 05, 2007 4:39:09 PM.* **Status:** *Ready*
Web_API_Server_Java_750_AlexR; aspirin : 2020;
Backup config server; aspirin : 2040;
Start time: *Monday, March 05, 2007 4:38:50 PM,* **Last request time:** *Not defined*, **Requests count:** *0*
Copyright (c) 2001-2007 Genesys Telecommunications Laboratories,Inc. All rights reserved.

LoadBalancing servlet configuration

Data from CME						Realtime data			
Service type	Name in CME	Host	Port	State	Version	Alias	Activity state	Times used	Time mark
CFGStatServer	Stat_Server_750	aspirin	4060	enabled	7.5.000.15	104	not_active	0	–
CFGWebAPIServer	Web_API_Server_Java_750_AlexR	alexr	80	enabled	7.5.000.00	138	active:P	0	–
CFGContactServer	ContactServer	aspirin	4400, ucsapi=4410	enabled	7.5.000.15	107	active:P	0	–
CFGInteractionServer	InteractionServer	aspirin	4420	enabled	7.5.000.10	108	active:P	0	–
CFGChatServer	ChatServer	aspirin	4800, webapi=8778	enabled	7.5.000.09	115	active:P	0	–
CFGEmailServer	E-MailServer	aspirin	4600, webapi=8777	enabled	7.5.000.10	114	active:P	0	–

Figure 30: 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 263. Figure 31 shows a successful load of the resulting test page.


Diagnostic information - Windows Internet Explorer
http://localhost:2746/DotNetSamples/TestTool750/LBTest.aspx
Diagnostic information
Page
Tools
Done
Local intranet
100%

SERVICE's information

LoadBalancer , Version 7.5.0.3
Diagnostic time: *Wednesday, February 28, 2007 3:16:57 PM.* **Status:** *Initialized, connected to Configuration Server, connected to SCS, connected to LCA*
Web_API_Server_NET_750_AlexR; aspirin : 2020;
Backup config server; aspirin : 2040;
Start time: *Wednesday, February 28, 2007 3:16:48 PM,* **Last request time:** *Wednesday, February 28, 2007 3:16:57 PM*, **Requests count:** *0*
Copyright (c) 2001-2007 Genesys Telecommunications Laboratories,Inc. All rights reserved.

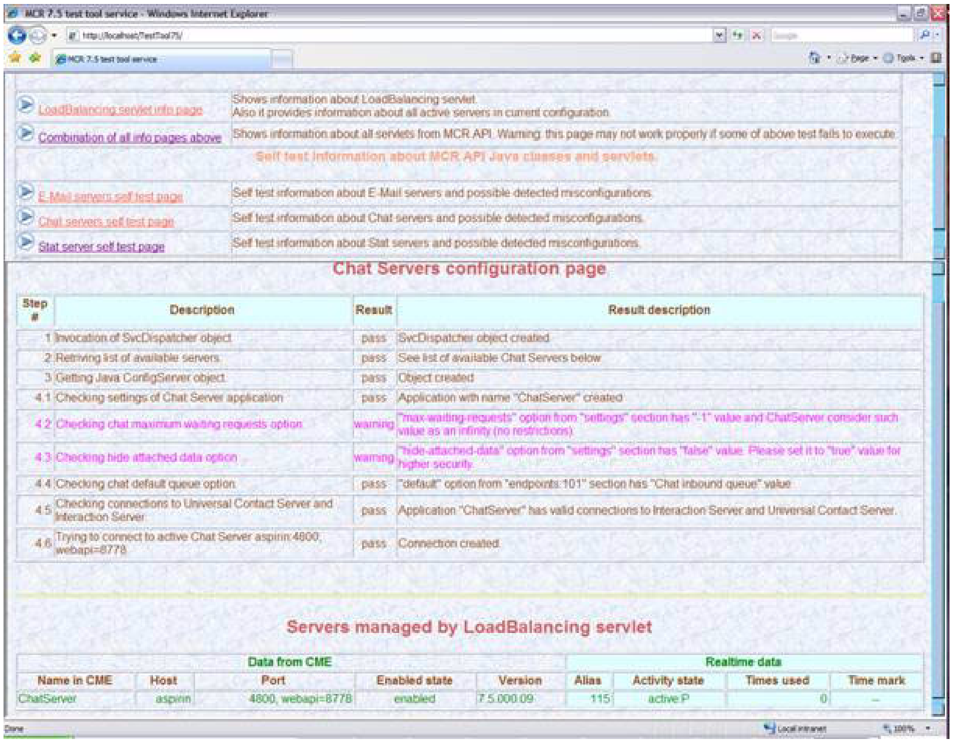
LoadBalancing SERVICE configuration

Data from CME							Realtime data			
Service type	Name in CME	Host	Ports	Tenants	State	Version	Alias	Activity state	Times used	Time mark
ContactServer	ContactServer	aspirin	4400; ucsapi=4410	mm750	enabled	7.5.000.15	107	active:P	0	–
InteractionServer	InteractionServer	aspirin	4420	mm750	enabled	7.5.000.10	108	active:P	0	–
SCS	SolutionControlServer_750	aspirin	4050	No tenants	enabled	7.5.000.07	102	active:P	0	–
ConfigServer	confserv	aspirin	2020	No tenants	enabled	7.5	99	active:P	0	–
StatServer	Stat_Server_750	aspirin	4060	mm750	enabled	7.5.000.15	104	active:P	0	–
EmailServer	E-MailServer	aspirin	4600; webapi=8777	mm750	enabled	7.5.000.10	114	active:P	0	–
MessageServer	MessageServer	aspirin	4080	No tenants	enabled	7.5.000.01	122	active:P	0	–
UniversalCallbackServer	CallbackServer652_wclunar1	wclunar1	6522	No tenants	enabled	6.5.2	143	not_active	0	–
ChatServer	ChatServer	aspirin	4800; webapi=8778	mm750	enabled	7.5.000.09	115	active:P	0	–

Figure 31: 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 32 shows a successful test for the Chat Server configuration.



MCR 7.5 test tool service - Windows Internet Explorer
http://localhost/TestTool75/
MCR 7.5 test tool service
Page
Tools
Done
Local intranet
100%

LoadBalancing servlet info page	Shows information about LoadBalancing servlet. Also it provides information about all active servers in current configuration.
Combination of all info pages above	Shows information about all servlets from MCR API. Warning: this page may not work properly if some of above test fails to execute.

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.

Chat Servers configuration page

Step #	Description	Result	Result description
1	Invocation of SvcDispatcher object	pass	SvcDispatcher object created
2	Retriving list of available servers	pass	See list of available Chat Servers below
3	Getting Java ConfigServer object	pass	Object created
4.1	Checking settings of Chat Server application	pass	Application with name "ChatServer" created
4.2	Checking chat maximum waiting requests option	warning	"max-waiting-requests" option from "settings" section has "-1" value and ChatServer consider such value as an infinity (no restrictions)
4.3	Checking hide attached data option	warning	"hide-attached-data" option from "settings" section has "false" value. Please set it to "true" value for higher security.
4.4	Checking chat default queue option	pass	"default" option from "endpoints:101" section has "Chat inbound queue" value
4.5	Checking connections to Universal Contact Server and Interaction Server	pass	Application "ChatServer" has valid connections to Interaction Server and Universal Contact Server.
4.6	Trying to connect to active Chat Server aspirin:4800, webapi=8778	pass	Connection created.

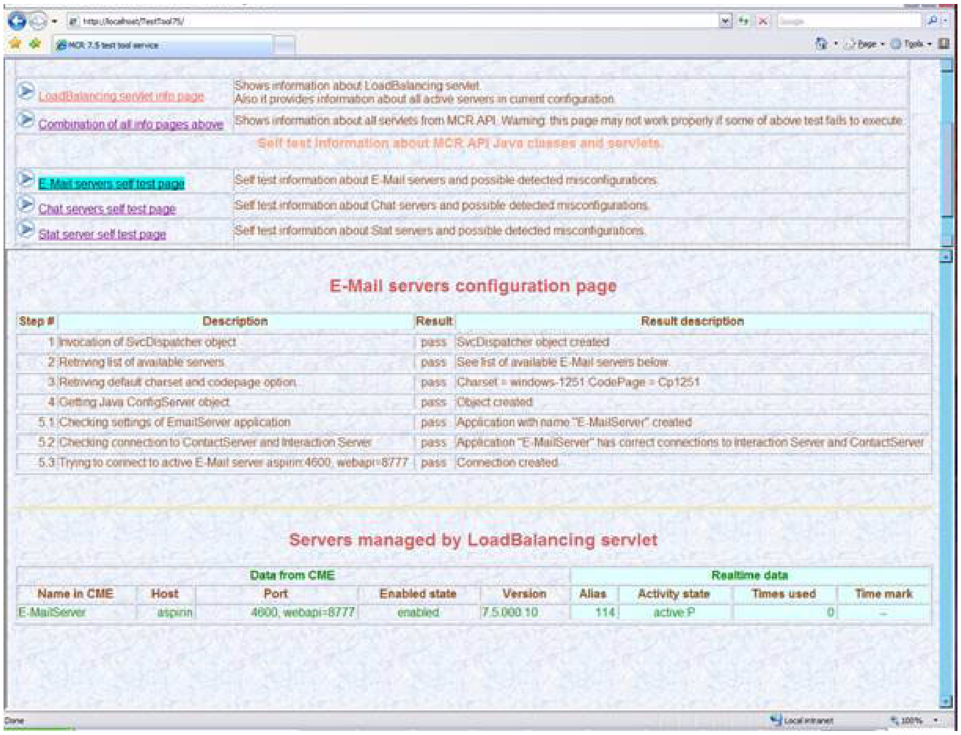
Servers managed by LoadBalancing servlet

Data from CME					Realtime data			
Name in CME	Host	Port	Enabled state	Version	Alias	Activity state	Times used	Time mark
ChatServer	aspirin	4800, webapi=8778	enabled	7.5.000.09	115	active P	0	--

Figure 32: Chat Server Configuration Page

Verifying the Configuration of E-Mail Server Java

To test the configuration of E-mail Server Java, scroll down in the upper frame of the Test Tools menu page and click the “E-mail servers self test page” link. Figure 33 shows a successful test for this configuration.



LoadBalancing servlet info page	Shows information about LoadBalancing servlet. Also it provides information about all active servers in current configuration.
Combination of all info pages above	Shows information about all servlets from MCR API. Warning: this page may not work properly if some of above test fails to execute.

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.

E-Mail servers configuration page

Step #	Description	Result	Result description
1	Invocation of SvcDispatcher object	pass	SvcDispatcher object created
2	Retrieving list of available servers	pass	See list of available E-Mail servers below
3	Retrieving default charset and codepage option	pass	Charset = windows-1251 CodePage = Cp1251
4	Getting Java ConfigServer object	pass	Object created
5.1	Checking settings of EmailServer application	pass	Application with name "E-MailServer" created
5.2	Checking connection to ContactServer and Interaction Server	pass	Application "E-MailServer" has correct connections to Interaction Server and ContactServer
5.3	Trying to connect to active E-Mail server aspirin:4600, webapi=8777	pass	Connection created

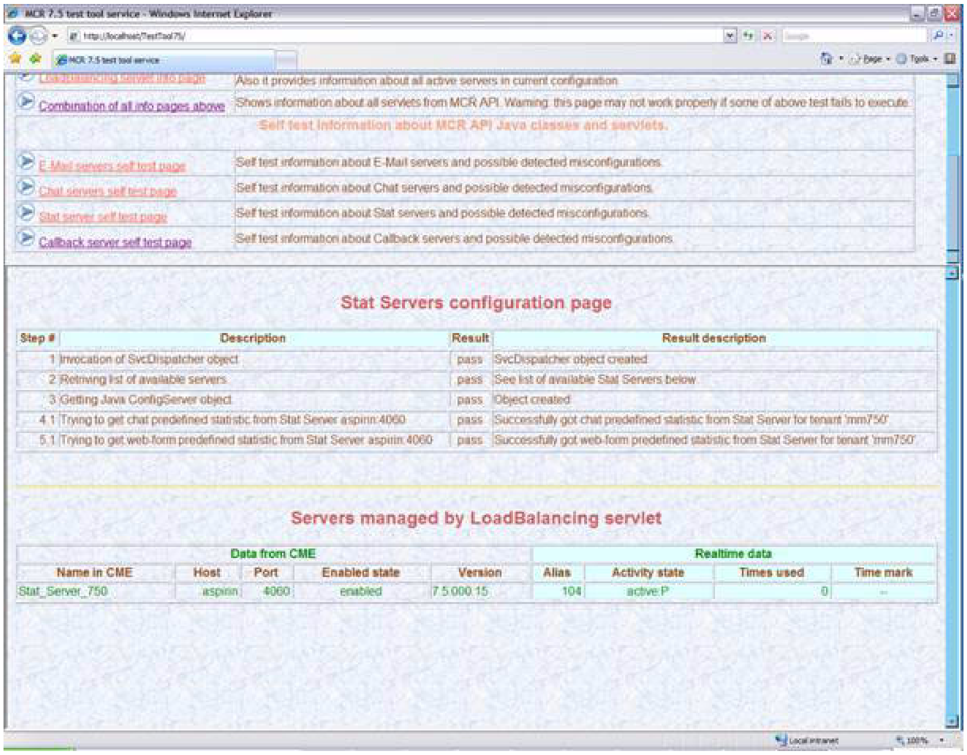
Servers managed by LoadBalancing servlet

Data from CME					Realtime data			
Name in CME	Host	Port	Enabled state	Version	Alias	Activity state	Times used	Time mark
E-MailServer	aspirin	4600, webapi=8777	enabled	7.5.000.10	114	active P	0	--

Figure 33: Configuration Page for E-Mail Server Java

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 34 shows a successful test for the Stat Server configuration.


MCR 7.5 test tool service - Windows Internet Explorer
http://localhost/TestTool75/
MCR 7.5 test tool service
Page
Tools
Local intranet
100%

LoadBalancing server info page	Also it provides information about all active servers in current configuration
Combination of all info pages above	Shows information about all servlets from MCR API. Warning: this page may not work properly if some of above test fails to execute

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.
Callback server self test page	Self test information about Callback servers and possible detected misconfigurations.

Stat Servers configuration page

Step #	Description	Result	Result description
1	Invocation of SvcDispatcher object	pass	SvcDispatcher object created
2	Retrieving list of available servers	pass	See list of available Stat Servers below
3	Getting Java ConfigServer object	pass	Object created
4.1	Trying to get chat predefined statistic from Stat Server aspirin:4060	pass	Successfully got chat predefined statistic from Stat Server for tenant 'mm750'
5.1	Trying to get web-form predefined statistic from Stat Server aspirin:4060	pass	Successfully got web-form predefined statistic from Stat Server for tenant 'mm750'

Servers managed by LoadBalancing servlet

Data from CME					Realtime data			
Name in CME	Host	Port	Enabled state	Version	Alias	Activity state	Times used	Time mark
Stat_Server_750	aspirin	4060	enabled	7.5.000.15	104	active:P	0	--

Figure 34: 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 *Multimedia 7.6 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

13 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

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 Multimedia 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 Multimedia media services.
- Scenario 2—How to assist website visitors (no user account) using Multimedia media services.
- Scenario 3—How to assist an on-site customer using Multimedia 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 Multimedia 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 Compound 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 272—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 Multimedia services can improve the overall user experience.

Index