
## Open Media Interaction SDK 7.6

# Services

# 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

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

## Technical Support from VARs

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

## Technical Support from Genesys

If you have purchased support directly from Genesys, please contact Genesys Technical Support at the 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)-118-974-7002 | 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:** 76sdk_dev_ixn_services-openmedia_03-2008_v7.6.001.00

# Table of Contents

# Preface

Welcome to the *Open Media Interaction SDK Services Developer's Guide*. This document introduces you to the concepts, terminology, and procedures relevant to the Open Media Interaction Service Libraries.

This document provides a high-level overview of Open Media Interaction SDK Service 7.6 features and functions, together with software-architecture information and deployment-planning materials.

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

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

This preface provides an overview of this document, identifies the primary audience, introduces document conventions, and lists related reference information:



The Open Media Interaction SDK (Software Development Kit) is built around the Media Interaction Layer library, which presents an API for developing third-party media applications. The library provides connectivity with Genesys Multi-Channel Routing (MCR) servers, so that your applications can create and manage Open Media interactions.Workforce Management database, serve real-time agent states, and provide browser-based functionality.

# Intended Audience

This guide is primarily intended for developers who are familiar with Simple Object Access Protocol (SOAP), Hypertext Transfer Protocol (HTTP), and XML (Extensible Markup Language) technologies. It assumes that you have a basic understanding of:

- Network design and operation.
- Your own network configurations.

You should also be familiar with these tools:

- XML Schemas
- SOAP (Simple Object Access Protocol)
- WSDL (Web Services Description Language)

Depending on the technology that you choose for client development, you might require a working knowledge of Java or of some other Web Services client-side programming language.

You should also be familiar with the Genesys Framework and with Genesys Multi-Channel Routing (MCR) 7.6 features.

# Usage Guidelines

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

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

The Genesys software functions available for development are clearly documented. No undocumented functionality is to be utilized without 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 opening chapter, this document contains the following chapters:

- Chapter 1, "About the Open Media Interaction SDK," on page 15. Introduces the Open Media Interaction SDK and its components, features, and scope of use.
- Chapter 2, "Connection," on page 21. Explains how to connect your application to GIS (the Genesys Interface Server).
- Chapter 3, "Data Transfer Object," on page 33. Introduces general DTO concepts.
- Chapter 4, "Events," on page 39. Introduces the event service.
- Chapter 5, "System Service," on page 55. Introduces the system service.
- Chapter 6, "Queued Interaction Layer," on page 61. Introduces the QIL (Queued Interaction Layer) service.
- Chapter 7, "Media Interaction Layer," on page 67. Introduces the MIL (Media Interaction Layer) service.

# 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:

76sdk_dev_ixn_services-openmedia_03-06_7.6.000.01

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:

- *Interaction SDK 7.6 Genesys Interface Server Deployment Guide,* which provides an overview of the Genesys Interface Server architecture and technologies and instructions for installing, configuring, starting and stopping, and uninstalling it.
- *Queued Interaction SDK 7.6 Java Developer's Guide,* which describes the features and capabilities of the QIL library that underlies the Open Media Interaction Services API.
- *Media Interaction SDK 7.6 Java Developer's Guide,* which describes the features and capabilities of the MIL library that underlies the Open Media Interaction Services API.
- *Open Media Interaction SDK 7.6 Services API Reference,* located on the Genesys documentation CD.
- *Genesys Agent Desktop 7.6 .NET Toolkit Developer's Guide*, which describes similar techniques and product features for developing .NET applications.
- The *Genesys Technical Publications Glossary,* which ships on the Genesys Documentation Library CD 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 CD, 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 *Genesys Technical Publications Glossary,* which ships on the Genesys Documentation Library CD 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 CD, 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.

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

- *Genesys 7 Supported Operating Systems and Databases*
- *Genesys 7 Supported Media Interfaces*

Genesys product documentation is available on the:

- Genesys Technical Support website at http://genesyslab.com/support.
- Genesys Developer website at http://devzone.genesyslab.com.
- Genesys Documentation Library CD, 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.

Chapter

# 1 About the Open Media Interaction SDK

This chapter introduces the Open Media Interaction SDK, its components, features, and scope of use.

The code snippets in this developer's guide are in C#, and there are differences due to the generated proxy and the language. For instance, in the *Open Media Interaction SDK Services API Reference for the .NET Proxy,* C# service interfaces are defined in accordance with the following rule: `I<service_name>Service`. In the generic *Open Media Interaction SDK 7.6 Services API Reference for the Java Proxy,* service interfaces are defined in accordance with the following rule: `<service_name>Service`

In this chapter, you will find the following topics:



## Features Overview

The Open Media Interaction SDK lets you build .NET applications to manage third-party media interactions and to monitor queues in the Genesys Framework.

The Open Media Interaction SDK presents a simple API for developing applications that:

- Access configuration information.
- Monitor queue activity.
- Manage third-party media interactions.

Your client-side applications use SOAP (Simple Object Access Protocol) or GSAP (Genesys Service Access Protocol) to communicate, through GIS (the Genesys Interface Server), to interact with the Genesys Framework. To develop successful client applications for Open Media Interaction SDK, you can:

- Use the Microsoft .NET Framework SDK, version 1.1 or 2.0, to create a C#-based application.
- Use the Apache AXIS toolkit, version 1.1 or version 1.3, to create stubs for a Java-based application.

# Components

The Open Media Interaction SDK Services product includes the following components.

- The *Open Media Interaction SDK 7.6 Services API Reference* in HTML and CHM formats, covering the Open Media Service API.
- This Developer's Guide, delivered on the documentation CD.

This set of components supports an application that allows you to manage third-party media interactions and monitor queues. It also provides such services as getting configuration information, managing UCS (Universal Contact Server), and so on.

The API is designed to allow development of applications that have specific requirements for the custom manipulation of particular service features. It is this assembly that directly communicates with GIS.

The Open Media Interaction SDK API Reference shows that the API comprises the following packages:

- `com.genesyslab.openmedia`—Exposes the main classes for connecting to GIS.
- `com.genesyslab.openmedia._event`—Exposes classes and interfaces related to event notification.
- `com.genesyslab.openmedia.soa`—Exposes open media services and their related classes.
- `com.genesyslab.soa`—Exposes additional container classes.

# Platform Requirements

The platform requirements for developing your application are a little different from the platform requirements for your final application in production.

## Development Platform

For .NET development, you need:

- Microsoft .NET Framework SDK, version 1.1 or version 2.0 (available at `http://msdn.microsoft.com/netframework/`).
- Microsoft Visual Studio .NET 2003 or 2005.

For Java development, you need:

- Apache AXIS toolkit, version 1.1 or version 1.3 (available at `http://xml.apache.org/axis/index.html`).
- Java Development Kit (JDK), version 1.3, 1.4.*x* or 1.5.

## Production Runtime Platform

For .NET development, you need:

- Microsoft .NET Framework version 1.1 or version 2.0 (available at `http://msdn.microsoft.com/netframework/`).

For Java development, you need:

- Apache AXIS toolkit, version 1.1 or version 1.3 (available at `http://xml.apache.org/axis/index.html`).
- Java Runtime Environment (JRE), version 1.3, 1.4.*x* or 1.5.

# Scope of Use

The Open Media Interaction SDK's typical usage scenarios include:

- Getting overall configuration information:
  - Application information.
  - Business attributes.
- Managing third-party media interactions:
  - Creating interactions of third-party media types.
  - Submitting a third-party interaction to Interaction Server.
  - Stopping the processing of third-party interactions in Interaction Server.
  - Managing callbacks from Interaction Server through ESP (External Service Protocol).
- Managing third-party interaction data in UCS (Universal Contact Server):
  - Saving third-party interaction data in UCS.
  - Updating third-party interaction data in UCS.
  - Finding third-party interaction data in UCS.
- Managing queues and queued interactions:
  - Starting the monitoring of a queue.

  - Stopping the monitoring of a queue.
  - Listening for changes to queues—status and interaction activity.
  - Listening for changes to interactions—status and properties.
- Managing connections to Genesys servers. Connection services involve the following components and protocols:
  - Interaction Server.
  - Configuration Layer.
  - UCS (Universal Contact Server).
  - ESP (External Service Protocol) to handle interactions' callbacks through the Interaction Server.

# Architecture

The Open Media Interaction SDK provides you with a service-oriented API that enables your application to connect to GIS. Through the services exposed in this API, you application can send requests to GIS remote services that manage these requests. GIS manages all connections to the Genesys Framework and Genesys servers, as shown in Figure 1 on page 18.

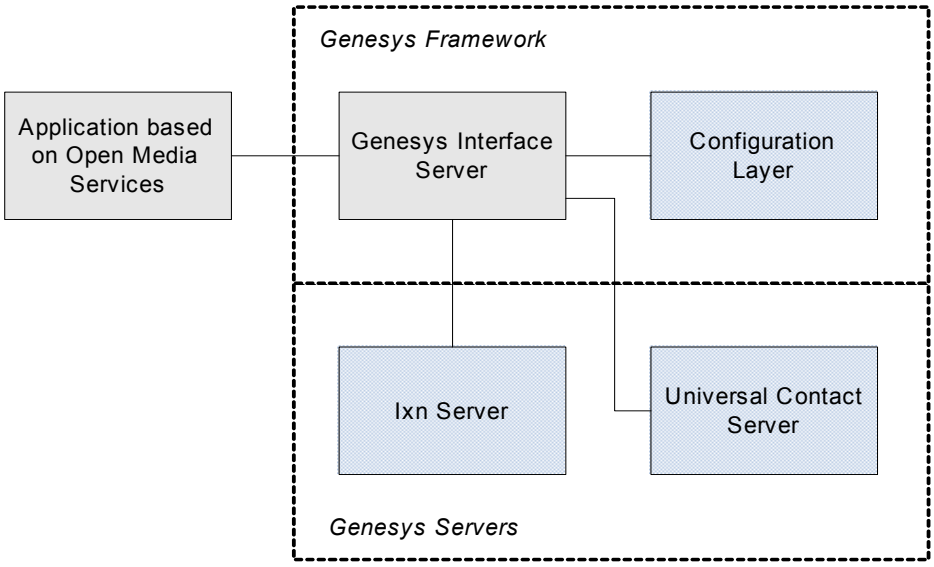


**Figure 1: Open Media Services Architectural Overview**

On the GIS (Genesys Interface Server) side, the exposed services deal with the Genesys Framework and perform the client-side services' requests.

## Service-Oriented Architecture

The Service-Oriented Architecture (SOA) is a specific type of distributed system in which features are exposed through services. When you are using the Open Media Interaction Service APIs, you are dealing with service interfaces

that do not manage anything locally. Each service defines a specific feature of your distributed system. Data management and actions are performed by GIS and you are concerned only with the interface descriptions.

## Multithreaded

The API is thread-safe and therefore your application can run in multithreaded environments. In particular, parallel threads can make calls to the same services' methods at the same time without blocking.

## Synchronization

Your application establishes a link with GIS, exposing the service that performs your client-application requests. The communication with GIS is synchronous.

## Connectivity

Connections to Genesys servers are maintained by GIS. Your client-side application can be notified of servers' statuses—namely, the loss of a connection.

For further information, refer to the *Interaction SDK 7.6 Genesys Interface Server Deployment Guide*.

### Framework Compatibility

GIS connects to the Configuration Layer of the Genesys Framework Suite. This server stores configuration information, such as application parameters, or objects' descriptions, such as business attributes and queues.

### MCR Compatibility

GIS connects to Genesys Multi-Channel Routing (MCR) and provides full multimedia support for third-party media and queued interactions. Connectivity is provided for the following MCR servers:

- MCR Interaction Server—This server manages interaction information with the Genesys Framework. GIS communicates with Interaction Server to manage Open Media interactions in queues. Additionally, Interaction Server can submit requests to the application integrating the library through External Service Protocol and GIS.
- MCR Universal Contact Server—This server manages contact-related information. Open media services use the UCS database to store third-party media interactions.

Chapter

# 2 Connection

This chapter explains how to connect your application to GIS (the Genesys Interface Server) with the Open Media Interaction SDK.

To communicate with GIS, your application uses one of the two following protocol:

- Genesys Service Access Protocol (GSAP)
- Simple Object Access Protocol (SOAP)

In this chapter, you will find the following topics:



## Generating a Java Proxy

You can use a toolkit to generate a proxy from the provided WSDL files—for example, Apache AXIS toolkit version 1.1 or version 1.3 for Java development (for further information, see: `http://ws.apache.org/axis/java/user-guide.html`).

In this case, use the GIS session service to connect your client application and to set options. Refer to the *Statistics SDK Developer's Guide* for further details about the session service, and see "</configuration>" on page 31 for further details about available options.

### Opening a Session

The first step your Open Media Interaction Service client application must perform is to open a session in GIS to get a session ID, which must be passed in the URL of any SOAP requests. As your application creates services, for

each service, specify the ENDPOINT_ADDRESS_PROPERTY and the session ID as shown in the following code snippet.

```
/// Creation of a system service using a stub created with
/// Apache Axis toolkit 1.1

import com.genesyslab.www.openmedia.*;
import com.genesyslab.www.openmedia.soa.*;
import com.genesyslab.www.openmedia.soa.openmedia.*;

//Creating a gis session - GIS server location set when
//generating the stub
SessionServiceServiceSoapBindingStub sessionService =
    (SessionServiceServiceSoapBindingStub) new
        SessionServiceServiceLocator().getSessionServiceService();

// Time out after a minute
sessionService.setTimeout(60000);
Identity id = new Identity();
id.setPrincipal("example");
id.setCredentials("");
sessionId = sessionService.login(id);

System.out.println("sessionId= " + sessionId);
sessionService._setProperty(
   sessionService. ENDPOINT_ADDRESS_PROPERTY,
   sessionService._getProperty(
                        sessionService.ENDPOINT_ADDRESS_PROPERTY)
   + "?GISsessionId=" + sessionId);

SystemServiceSoapBindingStub systemService =
   (SystemServiceSoapBindingStub)
     new SystemService_ServiceLocator().getSystemService();

systemService.setTimeout(60000);

/// Property used to pass session id in requests
systemService._setProperty(
   systemService.ENDPOINT_ADDRESS_PROPERTY,
   systemService._getProperty(
                           systemService.ENDPOINT_ADDRESS_PROPERTY)
   +"?GISsessionId=" + sessionId);

// then using systemService service is similar to C#
// getting a DTO
```

```
KeyValue[] values = systemService.getApplicationInfoDTO();
```

# Using the C# Proxy to Connect

You can use the provided .NET proxy to minimize session management tasks and to simplify service creation. This proxy corresponds to the `Genesyslab.soa.core.dll` and `Genesyslab.soa.OpenMedia.dll` files, available on the GIS Product CD.

## Service Factory

The `com.genesyslab.ail.ServiceFactory` class is the entry point of the .NET proxy. You must create a `ServiceFactory` object in order to connect. The connection can be synchonous or asynchronous, according to the method called:

- `ServiceFactory.createServiceFactory()`—At creation, the factory instance tries to connect synchronously to GIS. If the connection fails, it raises an exception.
- `ServiceFactory.asyncCreateServiceFactory()`—After the factory creation, the factory instance tries to connect asynchronously to GIS till connection succeeds or till the factory is released. To monitor the connection status, you must specify a `IServiceFactoryListener` listener at the factory creation.

When you create the factory (synchronously, or asynchronously), you must specify parameters to configure your connection:

- You can fill a `Hashtable` and pass it at the `ServiceFactory` creation. See "XML Configuration File for .NET" on page 24 for details about options.
- You can use an XML file to configure your `ServiceFactory` object; by default, this file is the `openmedia-configuration.xml` file.

In your XML configuration file, you must at least specify the `WebServicesFactory` mandatory attribute with its corresponding URL. See "XML Configuration File Example" on page 26. You can also define several optional attributes attached to this mandatory attribute.

**Note:** An `openmedia-configuration.xml` file is available on the GIS product CD in the `tools/` directory.

To access the available services, you create them by calling the `createService()` method of your instantiated factory, as shown in the following code snippet.

```
IXxxService service = (IXxxService) factory.createService(typeof(IXxxService));
```

# XML Configuration File for .NET

In your XML configuration file, or in the default `ail-configuration.xml` file, you must specify for the `factory` tag one of the following two attributes with their `url` option, according to the protocol used to communicate with GIS:

- For GSAP:
  - `PropFactory`—The factory name.
  - `url` option—The value is `prop://[Server address]:[Server port]`.
- For SOAP:
  - `WebServicesFactory`—The factory name.
  - `url` option—The value is `http://[Server Address]:[Server Port]/gis`.

This context can contain the options defined in the following subsections.

## Optional GSAP Attributes

Table 1 presents the GSAP optional attributes available for an XML configuration file written for the Open Media Interaction Services .NET proxy.

**Table 1: Optional GSAP Attributes**

| Name | Type | Description |
|---|---|---|
| `logger` | string | The path to the log file. |
| `logger.level` | string | The level of the `ROOT` logger. |
| `logger.levels` | string | The levels of the loggers. |
| `initial.connect.timeout` | string | The timeout interval for the first connection to the GSAP Connector in synchonous mode. |
| `timeout.ack` | string | The timeout interval for acknowledgements from the server, in milliseconds. |
| `timeout.response` | string | The timeout interval for responses from the server, in milliseconds. |
| `threads.max.worker` | string | Maximum number of threads in system pool. Should be greater than 50. |
| `threads.max.io` | string | Maximum number of threads for I/O operations in system pool. Should be greater than 50. |
| `connector.buffersize.receive` | string | Receive buffer size for the sockets operations, in bytes. Should be greater than 8000 bytes. |

**Table 1: Optional GSAP Attributes (Continued)**

| Name | Type | Description |
|---|---|---|
| `connector.buffersize.send` | string | Send buffer size for the sockets operations, in bytes. Should be greater than 8000 bytes. |
| `connector.tcpnodelay` | string | Should be `true`. Do not change this option. |

## XML Optional SOAP Attributes

Table 2 shows all the attributes that you can define for SOAP.

**Table 2: Optional SOAP Attributes**

| Name | Type | Description |
|---|---|---|
| `UseCookieContainer` | bool | Specifies whether or not the use of cookie containers is alloed. By default, it is set to `false`. You must set it to `true` to manage `http` sessions. This is mandatory for enabling high availability. |
| `BackupUrls` | string | A list of backup URLs to be used in case of disconnection, separated by commas as shown in this example: `"[http://[host1][:port1]/gis,http://[host2][:port2]/gis]"` |
| `Timeout` | int | The timeout interval for an XML web service client that waits for a synchronous XML web service request, to complete, in milliseconds. The default value is `100000` milliseconds. |
| `NbRetriesOnFailure` | string | The maximum number of reconnection attempts when calling a service method. The default value is `0`. |
| `RetryPeriodOnFailure` | string | The period in milliseconds between two reconnection attempts. |
| `ThreadPool.MaxWorkerThreads` | int | Indicates the maximum number of worker threads allowed at runtime. You must increase this number if your application makes multiple calls to service method, especially if the calls concern the `IEventService.getEvents` method. |

**Table 2: Optional SOAP Attributes (Continued)**

| Name | Type | Description |
|---|---|---|
| gis.asynchronousConnectionInterval | int | Specifies the time period in seconds (30 seconds by default) between two connection attempts. This option is used in case your application connects asynchonously. |
| gis.checkSessionInterval | int | The check session interval, in seconds. A value of 0 means no check. |
| gis.username | string | The GIS user name to log in the factory. Refer to Configuration Layer documentation for details. |
| gis.password | string | The GIS password to log in the factory. Refer to Configuration Layer documentation for details. |
| gis.tenant | string | The GIS tenant to use with the factory. Refer to Configuration Layer documentation for details. |
| gis.sessionId | string | The GIS session identity to use with the factory. If you use this option, do not use gis.username, gis.password, and gis.tenant. |
| notification.HTTPport | int | The notification HTTP port. The default value is 0, in which case the remote system chooses an open port on your behalf. |
| notification.createHTTPchannel | bool | Specifies whether to create an HTTP channel. The default value is true. |
| notification.objectURI | string | Specifies the remote object Universal Resource Identifier (URI). By default, the URI is generated by the WebServiceFactory. |
| notification.reachableURL | string | The reachable URI from the server. |
| service-point-manager.defaultConnectionLimit | int | The service point manager's connection limit. The default value is 2. |
| service-point-manager.maxServicePointIdleTime | int | The service point manager's maximum idle time. The default value is 900,000 milliseconds (15 minutes). |

## XML Configuration File Example

The following is an example of an XML configuration file for a SOAP connection:

```
<?xml version="1.0" ?>
```

```
<configuration default-factory="WebServicesFactory"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
   <factory name="PropFactory"
   classname="com.genesyslab.openmedia.propprotocol.PropFactory"
   assembly="OpenmediaServicesPropProtocol">
        <option name="Url" type="string" value="prop://[server host]:[server port]"/>
   </factory>
   <factory name="WebServicesFactory"
   classname="com.genesyslab.openmedia.WebServicesFactory"
   assembly="Genesyslab.Soa.OpenMedia">
     <option name="Url" value="http://frbred0059412:8080/gis"/>
     <option name="gis.username" value="default"/>
     <option name="gis.password" value="password"/>
     <!--
     <option name="gis.sessionId" value="1234567"/>
     <option name="gis.tenant" value=""/>

     <option name="timeout" type="int" value="100000"/>

     <option name="gis.checkSessionInterval" type="int" value="900"/>
     <option name="notification.HTTPport" type="int" value="0"/>
     <option name="notification.createHTTPchannel" type="bool" value="true"/>
     <option name="notification.objectURI" value="NotifLoad"/>
     <option name="notification.reachableURL"
     value="http://[client host]:[client port]"/>

     <option name="service-point-manager.defaultConnectionLimit" type="int" value="2"/>
     <option name="service-point-manager.maxServicePointIdleTime" type="int"
     value="900000"/>
     -->
   </factory>
</configuration>
```

# Using the Java Proxy to Connect

You can use the provided Java proxy to minimize session management tasks and to simplify service creation. This proxy is built from the Apache Axis toolkit version 1.3 and is available in the `tools/` directory on the GIS Product CD. It is composed of the fololwing `genesyslab-soa-core-proxy.jar` and `genesyslab-soa-open-media.jar` files.

This section presents how to connect to GIS, and how to use XML and options for instantiating this connection.

## Service Factory

The `com.genesyslab.soa.client.ServiceFactory` class is the entry point of the proxy. You must create a `ServiceFactory` object in order to connect. The

connection can be synchonous or asynchronous, according to the method called.

Except the default configuration file name, the process and the method to be called are identical to those described in "Service Factory" on page 23.

**Note:** The default XML configuration filename is `proxy-configuration.xml`. For further details, see "XML Configuration File for Java" on page 28.

To access the available services, you create them by calling the `createService()` method of your instantiated factory, as detailed in page 23.

# XML Configuration File for Java

In your XML configuration file, or in the default `proxy-configuration.xml` file, you must specify for the `factory` tag one of the following two attributes with their `url` option, according to the protocol used to communicate with GIS:

- SOAP
  - `AilWebServicesFactory`—The factory name.
  - `url` option—The value is `http://[Server Address]:[Server Port]/soa`.
- GSAP
  - `GSAPServiceFactoryImpl`—The factory name.
  - `url` option—The value is `prop://[Server Address]:[Server Port]`.

## XML Optional GSAP Attributes

Table 3 presents GSAP optional attributes available for an XML configuration file written for the Java proxy.

**Table 3: Optional GSAP Attributes**

| Name | rules | Description |
|---|---|---|
| `backupUrl` | string | Backup connection url to be used in case of disconnection. |
| `connect.interval` | positive integer | Interval between connection attempts in async mode and when reconnecting after connection loss (msec). |
| `connect.timeout` | positive integer | The timeout period in milliseconds for the TCP socket connection. |
| `timeout.ack` | positive integer | The timeout period to get an acknowledgement from the GIS, in milliseconds. If the timeout expires, the associated request fails. |

**Table 3: Optional GSAP Attributes (Continued)**

| Name | rules | Description |
|---|---|---|
| `timeout.response` | positive integer | The timeout period to get a response from the server, in milliseconds. If the timeout expires, the associated request fails. |
| `connector.buffersize.receive` | positive integer | Receive buffer size for the sockets operations, in bytes. Should be greater than 8000 bytes. |
| `connector.buffersize.send` | positive integer | Send buffer size for the sockets operations, in bytes. Should be greater than 8000 bytes. |
| `connector.tcpnodelay` | bool | Disables the Nagle's algorithm. Should always be `true`. Do not change this option. |

## XML Optional SOAP Attributes

Table 4 shows all the attributes that you can define for SOAP protocol.

**Table 4: Optional SOAP Attributes**

| Name | Description |
|---|---|
| `Username` | Username pour basic authentification. |
| `Password` | Password pour basic authentification. |
| `MaintainSession` | Indicates whether or not the HTTP session must be maintained. By default, it is set to `false`. |
| `DocumentMode` | Indicates the document mode, `false` for `rpc/encoding`, otherwise `true` for `document/literal`. The default value is `false`. |
| `NbRetriesOnFailure` | The maximum number of reconnection attempts when calling a service method. The default value is `0`. |
| `RetryPeriodOnFailure` | The period in milliseconds between two reconnection attempts. |
| `Connection.Timeout` | The timeout interval for an XML web service client that waits for a synchronous XML web service request to complete, in milliseconds. The default value is `100000` milliseconds. |
| `gis.asynchronousConnectionInterval` | Specifies the time period, in seconds (`30` seconds by default), between two connection attempts. This option is used if your application connects asynchonously. |

**Table 4: Optional SOAP Attributes (Continued)**

| Name | Description |
|---|---|
| gis.checkSessionInterval | The check session interval, in seconds. A value of 0 means no check. |
| gis.username | The GIS user name to log in the factory. Refer to Configuration Layer documentation for details. |
| gis.password | The GIS password to log in the factory. Refer to Configuration Layer documentation for details. |
| gis.tenant | The GIS tenant to use with the factory. Refer to Configuration Layer documentation for details. |
| gis.sessionId | The GIS session identity to use with the factory. If you use this option, do not use gis.username, gis.password, and gis.tenant. |
| notification.HTTPport | The notification HTTP port. The default value is 0, in which case the remote system chooses an open port on your behalf. |
| notification.reachableURL | The reachable URI from the server. http://[client host]:[client port] |
| http.proxyHost | The name for the proxy host. |
| http.proxyPort | The port of the proxy host. |
| http.proxyUser | The username for the proxy host. |
| http.proxyPassword | The password for the proxy host. |

## XML Configuration File Example for the Java Proxy

The following code snippet presents a proxy-configuration.xml file to be used with the Open Media Interaction Service Proxy Library for Java:

```
<?xml version="1.0" ?>
<configuration default-factory="AilWebServicesFactory"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
<factory name="AilWebServicesFactory"
   classname="com.genesyslab.ail.ws.client.AilWebServicesFactory" >
<option name="Url" value="http://localhost:8080/soa"/>
<option name="gis.username" value="default"/>
<option name="gis.password" value="password"/>
<option name="gis.tenant" value=""/>
<option name="MaintainSession" value="false"/> // maintain http session.
```

```
<option name="DocumentMode" value="false"/>
   //By default, document mode is false (for rpc/encoding); if true, document/literal
<option name="Username" value=""/> // username pour basic authentification
<option name="Password" value=""/> // password for basic authentification
<option name="http.proxyHost" value=""/> // proxy host
<option name="http.proxyPort" value=""/> // proxy port
<option name="http.proxyUser" value=""/>  // proxy user
<option name="http.proxyPassword" value=""/> // proxy password
<option name="ConnectionTimeout" value="60"/> // timeout request response in s
<option name="gis.asynchronousConnectionInterval"  value="30"/>
<option name="gis.checkSessionInterval"  value="900"/>
<option name="gis.sessionId" value="1234567"/>
<option name="notification.HTTPport" value="0"/>
<option name="notification.reachableURL" value="http://[client host]:[client port]"/>
    </factory>
</configuration>
```

Chapter

# 3 Data Transfer Object

This chapter introduces general DTO concepts, in the following sections:



## Introduction

In a client application, a transaction might require multiple server requests to complete. These requests and their responses require a significant amount of time to complete the transaction.

To improve the performance of a set of requests, the Open Media Interaction SDK's solution is to package all the required data into a Data Transfer Object (DTO) that can be sent in a single call.

A DTO is a generic container for a key-value list of data associated with several distinct remote objects. In the list, you specify only the keys that you are interested in. You use this list to retrieve or modify attributes' values according to their properties.

## DTOs in the Services API

The Open Media Interaction Service Layer makes use of the DTO Pattern for services' attributes that can be retrieved, set, or published. DTOs are involved in published events, in services' methods calls, and so on.

DTOs carry key-value attributes of several services. They are handled with dedicated methods and classes, as presented in the following sections.

# Dedicated Classes

Each service includes classes that gather attributes for DTO handling. These classes are named `*DTO`, and each class manages the DTO corresponding to its name prefix. For example, `QILInteractionDTO` is a class that manages the DTO of a QIL interaction.

The attributes list of a DTO is a `KeyValue` array. The `KeyValue` class is a very simple container that has two fields:

- `KeyValue.key`—the attribute name.
- `KeyValue.value`—the object corresponding to the attribute value.

# Attributes

Each service handles a set of objects, and proposes several domains defining available attributes to deal with objects' data.

**Note:** To determine what the available attributes of a service are, see the service interface description in the *Open Media Interaction SDK 7.6 Services API Reference*.

For example, the `ISystemService` interface is a service that deals with configuration information. It includes three defined domains:

- `application-info` defines common data associated with the application in the Configuration Layer.
- `business-attribute` describes some Open Media business attributes available in the Configuration Layer.
- `business-attribute-value` describes values for Open Media business attributes available in the Configuration Layer.

## Notation

For each service, the domain attributes used in DTO are defined in accord with the following rule:
*`domain`*`[[.`*`subdomain`*`]...]:`*`attributeName`*

For example, the following attributes exist:

- `mil-interaction:interactionId`
- `mil-interaction.ucs:parentId`

**Warning!** When you use an attribute, you must use the complete attribute name, including the domain and subdomains.

## Properties

Table 5 shows the properties that can be defined for an attribute.

### Table 5: Attribute Properties

| Attribute | Properties |
|---|---|
| `read` | The `I*Service` attribute is readable and can be retrieved with a `I*Service.get**DTO()` method. |
| `read-default` | The `I*Service` attribute is likely to be read often, so it is part of the default attributes. |
| `write` | The `I*Service` attribute is writable using a `I*Service.set**DTO()` method. |
| `event` | The attribute can be published via the event service. |
| `event-default` | The attribute is likely to be published often via the event service, so it is part of the default attributes. |

**Note:** To determine an attribute's properties, see the attribute description in the service interface definition in the *Open Media Interaction SDK 7.6 Services API Reference*.

## Wildcards

You can use the wildcards defined in Table 6 to access an entire set of attributes.

### Table 6: Attribute Wildcards

| Wildcards | Meaning |
|---|---|
| `*` | All the attributes of all the domains and subdomains. |
| `default` | All the attributes marked as `default` in all domains and subdomains. |
| `domain:*` | All the attributes of this domain. |
| `domain:default` | All the attributes marked as `default` in this domain. |
| `domain.*:*` | All the attributes of the subdomains. |

**Table 6: Attribute Wildcards (Continued)**

| Wildcards | Meaning |
|---|---|
| `domain.*:default` | All the attributes marked as `default` in the subdomains. |
| `domain.sub-domain:*` | All the attributes of this `domain.subdomain`. |
| `domain.sub-domain:default` | All the attributes marked as `default` in this `domain.subdomain`. |

# DTO Handling

Your application can use DTOs to read and write service attributes. DTOs also play an essential role in event handling.

**Note:** Genesys recommends that you avoid the use of wildcards in DTOs since they cause longer processing times for transactions.

This section shows how to read and set DTO attributes' values, and introduces the use of DTOs in events.

## Reading DTOs

Reading a DTO consists of reading a list of attributes identified with a `read` property in the service domain. You first define the list of attribute names, then use the appropriate `get*DTO()` method.

For example, if you want to read the `queue:status` and `queue:isMonitored` attributes for a set of queues, you will use the `IQILService.getQueuesDTO()` method. First, you must define an array of the attribute names to be read:

```
string[] myAttributeNames = new string[]{"queue:status",
                                         "queue:isMonitored" };
```

To retrieve the corresponding attribute values, call the `IQILService.getQueuesDTO()` method, as illustrated in the following code snippet:

```
/// Defining the list of queues you are interested in:
string[] myQueueIds = new string[]{ "queue0", "queue1"};

/// Retrieving for each queue the values for attributes
/// defined in mAttributeNames
```

```
QueueDTO[] myValues =
   myQueueService.getQueuesDTO(myQueueIds, myAttributeNames);
```

Each returned `QueueDTO` instance contains a key-value attribute array in the `QueueDTO.data` field. The following code snippet displays the attribute values in the data field of the `QueueDTO` object.

```
/// Displaying the ID of the first QueueDTO item returned
System.Console.WriteLine("Queue ID: "+myValues[0].queueId);

/// Displaying attributes' name and value:
foreach(KeyValue data in myValues[0].data )
{
   System.Console.WriteLine("{0}={1}",
                            data.key,
                            data.value.ToString());
}
```

The text displayed can be, for instance:

```
Queue ID: queue0
queue:status=ACTIVE
queue:isMonitored=false
```

## Getting DTOs in Events

Domain attributes that have the `event` property or the `event-default` property are published in events.

To determine which attributes are published by an event, refer to the *Open Media Interaction SDK 7.6 Services API Reference*. The available attributes are listed in the event description (part of the service interface definition).

When you subscribe to events, you specify which attributes must be retrieved. Then, the incoming events contain the `KeyValue` array with these attributes and their current values.

For further information about DTOs and event handling, see Chapter 4, “Events,” on page 39.

Chapter

# 4 Events

The event service is the `IEventService` interface defined in the `com.genesyslab.openmedia.soa` namespace. To manage events, your application must integrate this interface and use classes of the `com.genesyslab.soa._event` namespace to deal with it. This chapter is divided into the following sections:



## Introduction

Event handling is done through the event service and is based on the Subscribe/Publish Pattern. To deal with events, your application integrates the event service, which is in charge of all published events.

To be able to receive events, your application first subscribes to a list of events by calling the methods of the event service. Then, if your application performs modifications using the other Open Media services, or if the Genesys environment changes, the event service receives the corresponding events, as shown in Figure 2 on page 40.

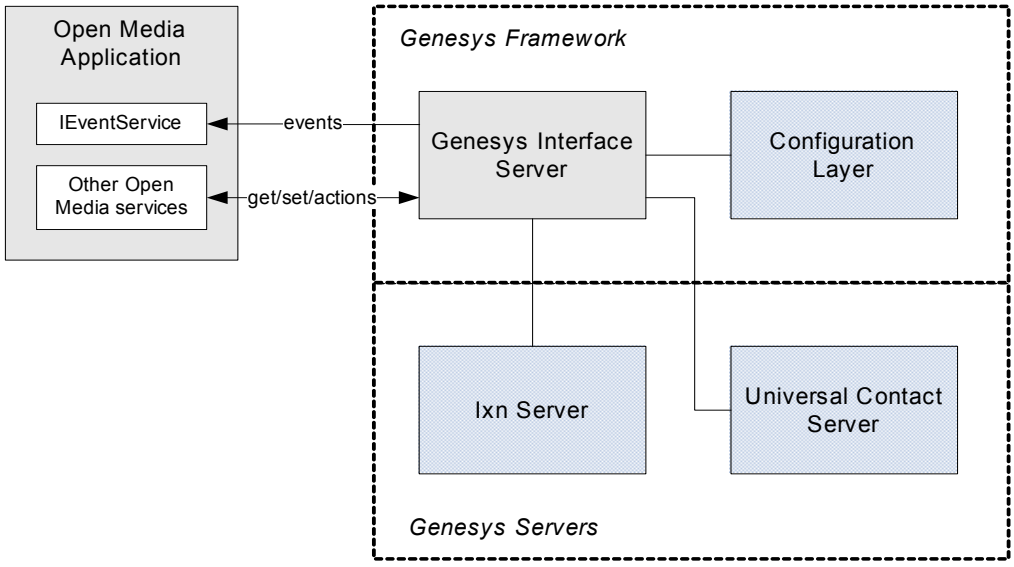


**Figure 2: The Integrated Event Service After Event Subscription**

Figure 2 shows that the event service is dedicated to getting events from GIS once the application has subscribed to some of them.

Notice that the other Open Media services do not provide any management related to events. Your application uses these services for performing actions through GIS on Genesys servers, or for accessing information from all the servers connected to GIS.

An event is specific to, and propagates a change in some data managed by, a particular service. This change can occur due to a modification in the Genesys environment. For example, if the content of a queue changes—that is, interactions are added or deleted—the event service can get a `QueueEvent` event that contains attributes reflecting these content changes.

For each service, the associated event names are listed in the interface description. For each type of event, you can see the list of available attributes to retrieve with the received event. You define the attribute to propagate with the event when your application subscribes.

# Understanding the Event Service

The event service has been designed to optimize network activity. Once you have subscribed to the events of a set of services, you get all the events in a single request—in either push or pull mode.

The following subsections introduce the main concepts of the event service, and of the classes of the `com.genesyslab.soa._event` namespace, that you should take into account in your application design.

# Events Associated with Services

As presented in "Introduction" on page 39, the event service is the only service integrated into your application that deals directly with events. All the events are received by the event service.

However, the other services are interfaces for a set of objects. Events can occur on the objects hidden by a service. Therefore, each service has its own set of events, which are designed to be appropriate to activity for that service.

To find the list of events for any particular service in the *Open Media Interaction SDK 7.6 Services API Reference*, open its service interface. For example, under `com.genesyslab.openmedia.soa`, open the `IQILService` interface, scroll past its list of attributes (in `domain:attribute` notation) to find the available types of events:

- `QueueEvent`
- `InteractionEvent`

For each service, the attributes that have an `event` property are likely to be published in the service events. Event descriptions in the *Open Media Interaction SDK 7.6 Services API Reference* list all the attributes published by each event.

# Understanding 'Topics' Objects

To manage events through the event service, you must define "topics" objects for creating or modifying the event subscription. The `IEventService` interface offers a set of features to dynamically remove, add, or modify subscriptions, according to your application needs.

The "topics" objects describe which event to subscribe or remove for each service. For each event subscribed, these objects describe which published attributes to retrieve. The "topics"-related interfaces are the following:

- For subscription: `TopicsService`, `TopicsEvent`, and `Topic`.
- For modifying the subscribed events: `TopicsServiceRemove`, `TopicsEventRemove`, and `Topic`.

The `TopicsService` interface lets your application subscribe to the events of a particular service. Each `TopicsService` instance associates a set of `TopicsEvent` with an Open Media service, as presented in Figure 3.

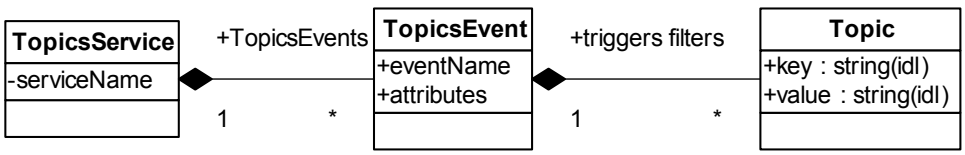


**Figure 3: The TopicsService Class Diagram**

Your application should subscribe to general `TopicsService` objects for every service that your application integrates. The `TopicsEvent` objects associated with a `TopicsService` define each event type you want to subscribe, by using:

- Triggers to define which specific events you want to receive.
- Attribute keys list to retrieve services' attribute values with the event.

**Note:** In the `TopicsEvent` object, you might notice the `filters` field. This field is for future releases. You do not need to take it into consideration.

The following subsections explain these aspects of event handling.

## Understanding Triggers

Triggers identify the Genesys object related to an event.

For example, if your application uses the QIL service to monitor queues, your application can subscribe to `QueueEvent`. Your application must specify a trigger, that is, which queue to monitor. For example, your application would specify `queue0` to receive any `QueueEvent` concerning `queue0`.

Triggers are values or fields of some published attributes listed in event descriptions of the *Open Media Interaction SDK 7.6 Services API Reference.* An event matches a `TopicsEvent` if its published attributes match one of the triggers.

The following sections present the general matching process for triggers and filters.

## Retrieved Events and TopicsEvents

Whatever the type of event received on the server-side application, the `IEventService` interface retrieves only `Event` objects.

The `Event` class is part of the `com.genesyslab.ail.ws._event` namespace. Its attributes include the following:

- `eventName`—a string specifying the event type.
- `serviceName`—a string specifying the service name related to the event.
- `triggers`—a key-value array of the triggers matched by the event.
- `attributes`—a key-value array of the published attributes propagated with the event.

In each description of events in the *Open Media Interaction SDK 7.6 Services API Reference*, the published attributes are listed. Only these attributes can be propagated in the `Event.attributes` field.

The `TopicsEvent` class lets your application specify the keys of the published attributes to retrieve with an `Event`.

Figure 4 illustrates the relationship between the attribute keys of a `TopicsEvent`, the published attributes of an event, and the key-value pairs propagated with an `Event` object.

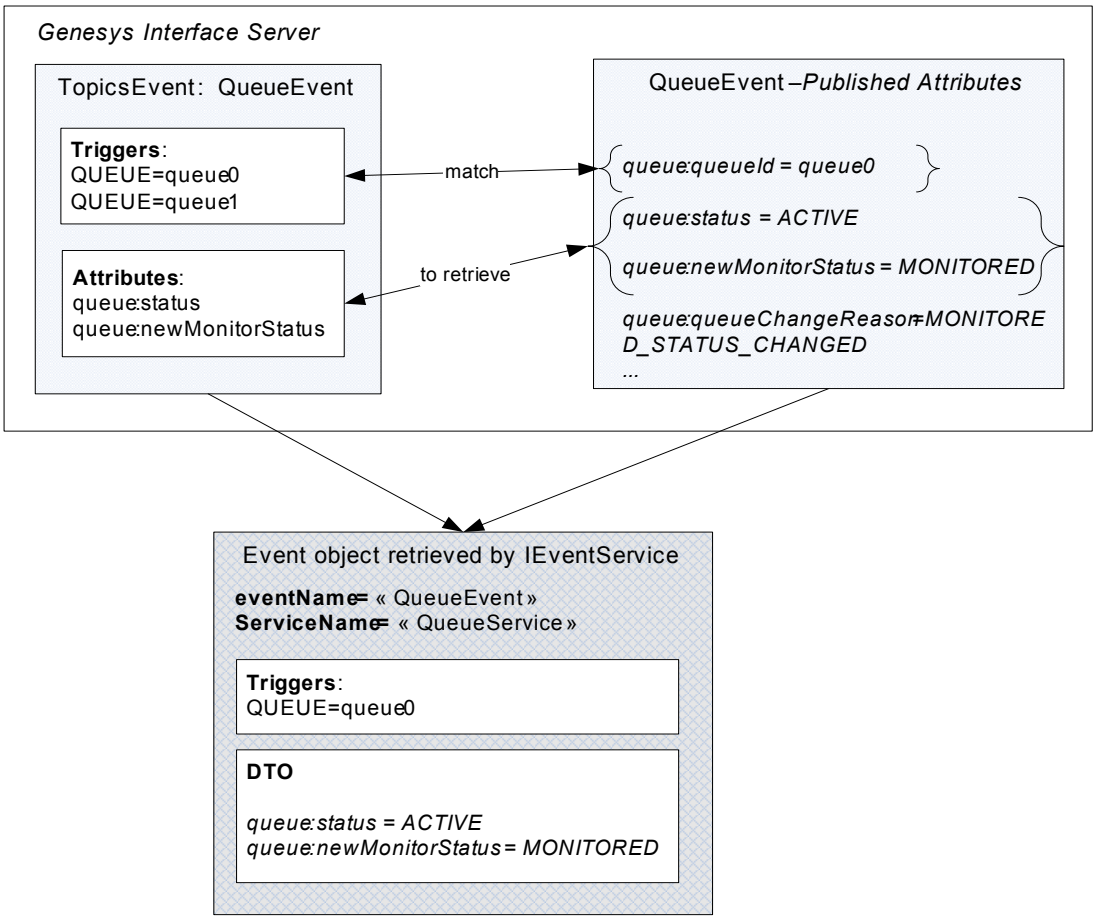


**Figure 4: TopicsEvent and Event Relationship**

As shown in Figure 4, the attribute keys specified in the `TopicsEvent` determine which attributes are propagated in the `Event` object retrieved by the `IEventService` from GIS.

# Handling Subscription and Topics

According to your requirements, your application must deal with Open Media service events—for example, to monitor queues and service back ends. Topics objects give you the opportunity to fine-tune your application during runtime.

This section assumes that you have read “Understanding the Event Service” on page 40.

This section details how to build topics objects and subscribe to events, how to modify topics objects so that you can get more or less information with events, and how to remove topics objects so that you no longer receive the associated events.

Therefore, this section is divided into the following subsections:

- "Creating TopicsService and TopicsEvent" on page 44.
- "Subscribing to the Events of a Service" on page 46.
- "Handling Subscription Errors" on page 49.

# Creating TopicsService and TopicsEvent

As explained in "Understanding 'Topics' Objects" on page 41, your application must create `TopicsService` and `TopicsEvent` objects to subscribe to events.

## Creating a TopicsService Object

The `TopicsService` class associates a specific service with an array of `TopicsEvent` to which to subscribe (see Figure 3, "The TopicsService Class Diagram," on page 41).

The `TopicsService.TopicsEvents` array must contain `TopicsEvent` objects of events occurring for the `TopicsService.serviceName` service.

For example, `QueueEvent` and `InteractionEvent` might occur for the `QueueService` service if interactions and queues change in the Genesys Framework. They can be specified in the `TopicsEvent` fields of a `TopicsService` object dedicated to the queue service.

The following code snippet defines a `TopicsService` object for the queue service. Its `TopicsEvent` field lets your application subscribe to `QueueEvent` only.

```
/// Creating a TopicsService for the Queue Service
TopicsService myTopicsServices = new TopicsService();
myTopicsServices.serviceName = "QueueService";

/// Creating Topics Events for the Queue Service
TopicsEvent[] myTopicsEvents = new TopicsEvent[1] ;

/// Defining a topic event for QueueEvent
myTopicsEvents[0] = new TopicsEvent() ;
myTopicsEvents[0].eventName = "QueueEvent" ;
/// ...

/// Adding the previous TopicsEvents to the TopicsService object
myTopicsServices.topicsEvents = myTopicsEvents ;
```

**Note:** Refer to the *Open Media Interaction SDK 7.6 Services API Reference* for more information about available events: See the interface description of a service.

## Adding TopicsEvents

For each `TopicsService`, you must fill in the `TopicsService.TopicsEvents` field with `TopicsEvent` objects that define the events to which to subscribe, using:

- Triggers to determine which Genesys objects to listen to.
- The list of attributes to retrieve in a DTO when the targeted events occur.

In the previous subsection, the code snippet defines a `TopicsService` instance for the queue service. The following code snippet shows how to set triggers to a `TopicsEvent` instance, so that your application gets events of type `QueueEvent` for two queues named `queue0` and `queue1`.

```
/// Defining a topic event for QueueEvent
myTopicsEvents[0] = new TopicsEvent() ;
myTopicsEvents[0].eventName = "QueueEvent";

/// Defining triggers for queue0 and queue1
myTopicsEvent.triggers = new Topic[2] ;

/// Specifying the queues to be listened
myTopicsEvent.triggers[0]= new Topic();
myTopicsEvent.triggers[0].key = "QUEUE" ;
myTopicsEvent.triggers[0].value = "queue0";

/// Specifying the targeted queue
myTopicsEvent.triggers[1]= new Topic();
myTopicsEvent.triggers[1].key = "QUEUE" ;
myTopicsEvent.triggers[1].value = "queue1";
```

The above code snippet specifies that any `QueueEvent` occurring on `queue0` or `queue1` is retrieved by the event service.

Now you must specify which attributes to retrieve with these events. The attributes published within an event have an `event` property.

**Note:** For performance reasons, you should retrieve only the event attributes that your application will use, in order to minimize traffic with GIS and other Genesys servers.

The following code snippet completes the previous code snippets by setting a list of attributes to retrieve for `QueueEvent` events:

```
/// Setting the key list of attributes to retrieve in
/// QueueEvent events
myTopicsEvent.attributes = new String[] {
                           "queue:queueChangeReason",
                           "queue:status"
                           "queue:newMonitorStatus"} ;
```

Your application can use wildcards in the `TopicsEvent.attributes`, as presented in the following code snippet:

```
/// Retrieving all the queue attributes
myTopicsEvent.attributes = new String[] {"queue:*"};
/// ...
```

In the above code snippet, the asterisk wildcard specifies that all attributes in the queue domain having an `event` property are propagated. For further information about wildcards, see page 35.

For further information about the key-value pairs for triggers and published attributes, refer to the event descriptions in the *Open Media Interaction SDK 7.6 Services API Reference.*

# Subscribing to the Events of a Service

In the previous subsections, code snippets defined a `TopicsService` instance for `QueueEvent` events of the queue service. This section shows how to subscribe by:

- Creating an `IEventService` instance.
- Creating a subscriber and subscribing using topics.
- Subscribe to new topics and unsubscribing to topics.

## Getting an Event Service Instance

To get an `IEventService` instance, make a call to the `ServiceFactory.createService()` method, as shown here:

```
IEventService eventservice = (com.genesyslab.openmedia.soa.IEventService)
   factory.createService(typeof(com.genesyslab.openmedia.soa.IEventService));
```

## Creating a Subscriber

Creating a subscriber is essential when your application needs to get events with the event service. You use this subscriber to:

- Initially subscribe your application to a set of events.

- Update the list of events to which you subscribed, by subscribing to and unsubscribing from new topics.

You must use a single subscriber for your application to manage all subscription-related actions on the event service.

The following code snippet shows how to create a new subscriber for your application:

```
/// Creating a Subscriber
SubscriberResult mySubscriber = eventservice.createSubscriber(null,myTopicsServices) ;
```

**Note:** Use this `SubscriberResult` for your further subscribing and unsubscribing operations. This ensures the use of a single subscriber for your application.

## Further Subscribing

During runtime, your application's needs for event-propagated data can change. Your application can define new `TopicsService` objects and use the `IEventService.subscribeTopics()` method to subscribe, as presented in the following code snippet:

```
/// Creating the array of new topics
TopicsService[] newTopicsServices = new TopicsService[1] ;
///...
/// Subscribing
eventservice.subscribeTopics( mySubscriber.subscriberId,
                                newTopicsServices);
```

**Warning!** When your application subscribes to a `TopicService` using a `TopicsEvent` with a trigger that has already been subscribed, filters and attributes are all replaced by the new ones.

## Unsubscribing from Topics

Your application may unsubscribe from a `TopicsService`, or modify `TopicsEvents` content, during application runtime to fulfill your application's needs. The following subsections detail the corresponding `IEventService` features.

### Removing All the Topics Events

Your application can remove all the topics events for all the services. Use the `IEventService.unsubscribeAllTopics()` method.

The following code snippet unsubscribes from all the topics objects defined for your application subscriber:

```
eventservice.unsubscribeAllTopics(mySubscriber.subscriberId);
```

All the `TopicsEvent` previously defined with a `TopicsService` are removed. Your application no longer receives events.

### Removing Specific Topics for a Service

The process of removing a specific topic for a service is similar to the subscribing process. Instead of subscribing with a `TopicsService` array, your application unsubscribes with a `TopicsServiceRemove` array.

A `TopicServiceRemove` object is dedicated to a service and includes the `TopicsEventRemove` objects that list the removed events for this service. The removed events are associated with a trigger.

The following code snippet removes the trigger `queue0` of the `QueueEvent` for the queue service:

```
/// Defining the trigger
Topic myTriggerToRemove = new Topic();
myTriggerToRemove.key = "QUEUE";
myTriggerToRemove.value = "queue0";

/// Creating the array of events to remove
TopicsEventRemove[] myTopicsEventToRemove = new TopicsEventRemove[1];
myTopicsEventToRemove[0] = new TopicsEventRemove();

/// Setting the trigger for the Queue
myTopicsEventToRemove[0].eventName = "QueueEvent";
myTopicsEventToRemove[0].triggers = new Topic[1];
myTopicsEventToRemove[0].triggers[0] = new Topic();
myTopicsEventToRemove[0].triggers[0] = myTriggerToRemove;

/// Creating the array of TopicsServiceRemove
TopicsServiceRemove[] myTopicsServiceToRemove = new TopicsServiceRemove[1];

/// Creating a TopicsServiceRemove for the Queue Service
myTopicsServiceToRemove[0] = new TopicsServiceRemove();
myTopicsServiceToRemove[0].serviceName="QueueService";

/// Associating the previous topics to the Queue Service
myTopicsServiceToRemove[0].topicsEventsRemove = myTopicsEventToRemove;

/// Unsubscribing
myEventService.unsubscribeTopics( mySubscriber.subscriberId, myTopicsServiceToRemove);
```

The above code snippet ensures that the subsequent `QueueEvents` retrieved with the `mySubscriber.subscriberId` no longer include events for `queue0`.

## Handling Subscription Errors

When your application subscribes or unsubscribes, the topic objects are processed sequentially: if an error occurs for one topic, the remaining topics are processed. The occurred errors are returned in an array of `TopicServiceError` objects.

```
/// subscribing to topics
TopicServiceError[] myTopicsServiceErrors =
   eventservice.subscribeTopics( mySubscriber.subscriberId, myTopicsServices);

/// Displaying the topics errors
foreach(TopicServiceError err in myTopicsServiceErrors)
{
   System.Console.WriteLine("Subcr. error for event {0}: key = {1}
                            val = {2}",
                            err.eventName, err.filter.key,
                            err.filter.value.ToString());
}
```

In the above code snippet, the event service processes a subscription and errors are displayed in the console.

# Getting Events

There are two available modes to get events:

- Pull mode—your application retrieves the events.
- Push mode—your application is notified of the events.

## Pull Mode

In pull mode, your application must periodically retrieve events; it is not notified when an event occurs. The server-side application waits for the client-side application request to deliver the subscribed events.

### Retrieving Events

To retrieve events, your application defines topics for the services, then subscribes to these topics. See “Subscribing to the Events of a Service.”

Once your application has subscribed, it can retrieve events associated with the `SubscriberResult.subscriberId` identifier by calling the `IEventService.getEvents()` method.

The following code snippet is an example of a `getEvents()` call:

```
/// Retrieving the last occurred events
/// timeout in seconds is set to 1
Event[] events = myEventService.getEvents(mySubscriber.subscriberId,
                                          1);

/// Displaying the events
foreach(Event evt in events)
{
   System.Console.WriteLine("Occured {0}",evt.ToString());
}
```

**Warning!** If you set a non-zero value for the timeout parameter of the `IEventService.getEvents()` method, this method does not return until either events occur or the timeout is reached.

### Specifics

In pull mode, the subscriber must be sure to retrieve the events before the server-side timeout is reached.

**Warning!** The default timeout is 10 minutes. If no event has been retrieved within 10 minutes, the subscriber is removed.

## Push Mode

In push mode, your application is notified of the occurring events. Your application must:

1. Implement the `notifyEvents()` method of a class inheriting the `INotifyService` interface.
2. Subscribe to the event service.

Then, during runtime, whenever events occur, the `notifyEvents()` method is called and its code is executed.

### Using the INotifyService Interface

Your application must create a class inheriting the `com.genesyslab.ail.ws._event.INotifyService` class. This inherited class must implement the `INotifyService.notifyEvents()` method.

The following code snippet presents a short implementation of an inherited class with a `notifyEvents()` method that displays the content of notified events in the console:

```
public class NotificationImpl :
   com.genesyslab.soa._event.INotifyService
{
   public void notifyEvents(string subscriberId,
                     com.genesyslab.ail.ws._event.Event[] events)
   {
         if (events == null)
         {
            System.Console.WriteLine("notifyEvents - null \n");
            return ;
         }
         System.Console.WriteLine( "notifyEvents getEvents : " +
                                 events.Length + "\n") ;
         foreach( Event evt in events)
         {
            System.Console.WriteLine( "Service :"+ evt.serviceName
                                 + "Event: "+ evt.eventName
                                 + "timeStamp:"+ evt.timeStamp
                                 +"\n");
         }
   }
}
```

### Subscribing

Use an instance of your inherited `INotifyService` class to fill the `notif.notificationEndpoint` field.

```
Notification notif = new Notification();
notif.notificationEndpoint = new NotificationImpl();
/// Notif type for Web services
notif.notificationType="SOAP_HTTP";
```

Then, subscribe with the `Notification` instance.

```
SubscriberResult result =
   myEventService.createSubscriber(notif,myTopicsServices) ;
```

## Reading DTOs in Events

When your application subscribes to events, it specifies a set of published attributes to retrieve with the events (see “Creating TopicsService and TopicsEvent” on page 44).

The attributes can be accessed with the `Event.attributes` attribute, which is a `KeyValue` array. The following code snippet is a pull-mode example:

```
/// Retrieving the last occurred events
Event[] events = myEventService.getEvents(mySubscriber.subscriberId,
```

```
1);
foreach(Event evt in events)
{
   KeyValue[] attributes = evt.attributes ;
   foreach( KeyValue attr in attributes)
   {
      System.Console.WriteLine(
         "Service: {0}\tKey: {1} value: {1}",
         evt.serviceName, attr.key, attr.value) ;
   }
}
```

The above code snippet displays the attribute key-value pairs retrieved with the events.

# Event Notification in Java

This section describes how to use Open Media Interaction Services notification with Java. Several solutions are available to use unsolicited events in Java with GIS.

In this section, we use the Apache Axis SOAP toolkit to implement a client-side notification mechanism in a simple notification server.

This example supposes that we have GIS running on host *<GIS_HOST>* and port *<GIS_PORT>*. All the following subsections are related to this example.

## Notification Classes Generation

To generate classes used in notification events, we will use `WSDL2java`, a tool provided by Apache Axis. Replace the italicized placeholders when typing the following command line:

```
java org.apache.axis.wsdl.WSDL2Java
-o output
--server-side http://<GIS_HOST>:<GIS_PORT>/gis/services/OPENMEDIA_/NotifyService?wsdl
```

The required classes will be generated in the directory named *output*. These classes must be added to your source path. The `WSDL2java` tool generates a mapping file that maps the SOAP types to Java classes.

The tool generates the classes for each type from WSDL, using a type-mapping file (`deploy.wsdd`). It also generates the following server implementation class:

```
com/genesyslab/www/services/openmedia/wsdl/event/NotifyServiceSoapBindingImpl.java
```

This class has a method `notifyEvents(String subscriberId, Event[]events)`, which is called on each notification event, as shown in the following example:

```
public void notifyEvents(String subscriberId, Event[] events) throws
   java.rmi.RemoteException,
   com.genesyslab.www.services.ail.wsdl.event.WServiceException
{
   // Put action to process for each event received here
}
```

## Simple Notification Server

This subsection introduces the implementation of a simple notification server for your client application. To achieve this, you can use a little server provided by the Axis toolkit and identified as the following class:

```
org.apache.axis.transport.http.SimpleAxisServer
```

To provide this class with all the deployment information included in the `deploy.wsdd` file, start it as shown in the following code snippet:

```
org.apache.axis.client.AdminClient adminClient =
   new org.apache.axis.client.AdminClient();

String[] argsDeploy = {"deploy.wsdd", "-p", Integer.toString(port)};

adminClient.process(argsDeploy);
```

Once the server is started, you can browse `Notify Service` on the client side at:

```
http://client_host:client_port/axis/services/NotifyService?wsdl
```

When creating a subscriber in your application for the event service, you must define the notification location by setting the following fields to:

- `notificationEndPoint`
  `http://client_host:client_port/axis/services/NotifyService`
- `notificationType`
  `SOAP_HTTP`

Chapter

# 5 System Service

This chapter explains the `ISystemService` interface that provides information about configuration data and connection backends.

This chapter includes the following sections:



## Prerequisites

To follow the discussion in this chapter, you will need the *Open Media Interaction SDK 7.6 Services API Reference* (which is located in the `doc/` subdirectory of your Open Media Interaction SDK Services installation directory).

Discussion in this chapter also assumes that you read Chapter 2, "Connection," on page 21, Chapter 3, "Data Transfer Object," on page 33, and Chapter 4, "Events," on page 39 before starting reading this section.

## More System Essentials

This section introduces the system service and the essential information you need to know to integrate it to your application.

The system service provides configuration and system data through the attribute domains and methods listed in Table 7.

**Table 7: Domains and Related Methods**

| Domain | ISystemService Methods |
|---|---|
| application-info | getApplicationInfoDTO() |
| business-attribute<br>business-attribute-value | getBusinessAttributesDTO()<br>getMediaTypesDTO()<br>getInteractionTypesDTO()<br>getInteractionSubTypesDTO() |
| service-info | getServiceInfos() |

There are two main things you can do with the system service:

- **Get configuration data,** that is, application information, business attributes, and business attribute values.
- **Monitor connections to Genesys servers.**

The following sections detail how to make these steps stand out so that you can quickly learn to write your own real-world applications.

# Configuration Data

After your application has connected GIS, the configuration features of the system service give you access to the Configuration Layer data for this application's tenant, such as:

- Application information.
- Business attributes:
  - Media types defined in the `Media Type` section.
  - Interaction types defined in the `Interaction Type` section.
  - Interaction subtypes defined in the `Interaction Subtype` section.

## Getting Application Information

To get application information, you just call the `SystemService.getApplicationInfo()` method. It returns an array of key-value pairs, as shown in the following code snippet:

```
KeyValue[] applicationinfo =
   systemservice.getApplicationInfoDTO(new string[]{"application-info:*"});
```

# Getting Business Attributes

Business attributes and their values are configuration data available through the Configuration Layer. Business attributes define metadata for information concerning MIL and QIL interactions.

There are three types of business attributes that you can use in your open media application, defined in the `BusinessAttributeType` class:

- `MEDIA_TYPE`—Business attribute for media types.
- `INTERACTION_TYPE`—Business attribute for interaction types.
- `INTERACTION_SUBTYPE`—Business attribute for interaction subtypes.

A business attribute , available in the following interfaces:

- `BusinessAttributeDTO`—Metadata for a business attribute's type.
- `BusinessAttributeValueDTO`—A value for a business attribute's type.

The `BusinessAttributeDTO` interface defines metadata for information concerning MIL and QIL interactions.

To get a business attribute, use one of the `ISystemService` methods associated to the business-attribute domain (see Table 7 on page 56).

Each `BusinessAttributeDTO` interface contains a set of `BusinessAttributeValue` interfaces. Each `BusinessAttributeValue` describes a value characterized by the parent `BusinessAttribute`.

The following code snippet shows how to get `Media Types` business attributes and the associated values.

```
BusinessAttributeDTO businessAttributeDTO = systemservice.getMediaTypesDTO(
   new String[]{"business-attribute:displayName"},
   new String[]{"business-attribute-value:name","business-attribute-
   value:description"});
System.Console.WriteLine( businessAttributeDTO.name +" values are: " );
foreach(BusinessAttributeValueDTO val in businessAttributeDTO.values)
{
   String toDisplay = val.name+ ": " ;
   foreach(KeyValue pair in val.data)
   {
      toDisplay += pair.key + "="+ pair.value.ToString()+" ";
   }
   System.Console.WriteLine(toDisplay);
}
```

If your application handles QIL and MIL interactions, use business attributes to get more information about these interactions. In the `qil-interaction` and `mil-interaction` domains respectively of the `IQILService` and `IMILService`, you can notice that interactions are associated with values of business attributes.

For instance, if for a given `QILInteractionDTO`, the `qil-interaction:mediaType` is `fax`, you can get a `BusinessAttributeDTO` object for `Media Type`, as shown in the previous code snippet, and then, use it to get the `BusinessAttributeValueDTO` object containing data about the `fax` value.

Refer to the *Open Media Interaction SDK Services API Reference* for further details about classes.

# Monitoring Services

GIS maintains connections to the Genesys environment. Although you cannot manage these connections with the open media services, the system service lets your application access and monitor connections' status.

In this purpose, service features are available as `ServiceInfo` objects through the `ISystemService.getServiceInfosDTO()` method. Each `ServiceInfo` instance associates a `ServiceStatus` with a `ServiceType`.

Your application can access several types of services—see `ServiceType` for further details:

- `CONFIGURATION`—Connection to the Configuration Layer.
- `INTERACTION_SERVER`—Connection to Interaction Server.
- `UCS`—Connection to Universal Contact Server.
- `ESP`—Connection to Interaction Server using ESP (External Service Protocol.)

The following code snippet shows how to retrieve information for all services of your application.

```
ServiceInfo[] allServices = systemservice.getServiceInfos();
foreach(ServiceInfo service in allServices)
{
   System.Console.WriteLine(service.type.ToString() +" is in status " +
                            service.status.ToString() );
}
```

This code snippet can generate the following output:

```
CONFIGURATION is in status ON
INTERACTION_SERVER is in status ON
ESP is in status OFF
UCS is in status OFF
```

To monitor services, your application must subscribe to `ServiceEvent` and specify which service to monitor. There are four steps to complete:

1. Create a `TopicsService` for the system service.
2. Create a `TopicsEvent` for `ServiceEvent` events.

3. Add a trigger for each service to be monitored.
4. Subscribe.

The following code snippet shows how to subscribe to ServiceEvent events for connections to the Configuration Layer and Interaction Server.

```
/// 1. Creating a TopicsService for the System Service
TopicsService systemTopicsService = new TopicsService();
systemTopicsService.serviceName = "SystemService";
systemTopicsService.topicsEvents = new TopicsEvent[1] ;

/// 2. Defining a topic event for ServiceEvent
systemTopicsService.topicsEvents[0] = new TopicsEvent() ;
systemTopicsService.topicsEvents[0].eventName = "ServiceEvent" ;
//Setting attributes to retrieve
systemTopicsService.topicsEvents[0].attributes = new string[]{"service-info:info"};

/// 3. Adding triggers
systemTopicsService.topicsEvents[0].triggers = new Topic[2];
systemTopicsService.topicsEvents[0].triggers[0] = new Topic();
systemTopicsService.topicsEvents[0].triggers[0].key = "SERVICE_TYPE";
systemTopicsService.topicsEvents[0].triggers[0].value = "CONFIGURATION";

systemTopicsService.topicsEvents[0].triggers[1] = new Topic();
systemTopicsService.topicsEvents[0].triggers[1].key = "SERVICE_TYPE";
systemTopicsService.topicsEvents[0].triggers[1].value = "INTERACTION_SERVER";

///4. Subscribing with the subscriber ID of the application
TopicsService[] newTopics = new TopicsService[]{systemTopicsService};
eventservice.subscribeTopics( mySubscriber.subscriberId,newTopics);
```

Then, if any `ServiceEvent` occurs for the connection of the GIS to Configuration Layer or Interaction Server, your application will get an `Event` object through the event service.

In this `Event` object, you will get the new status for the connection service who caused the event.

```
Event[] events = eventservice.getEvents(mySubscriber.subscriberId,1);
foreach(Event evt in events)
{
   if (evt.serviceName == "SystemService" && evt.eventName== "ServiceEvent")
   {
      foreach (KeyValue att in evt.attributes)
      {
         //key is always service-info:info
         ServiceInfo srv = (ServiceInfo) att.value;
         System.Console.WriteLine(att.key+ ": "+ srv.type.ToString + " is in status"
                                  +srv.status.ToString());
      }
```

```
    }
}
```

Chapter

# 6 Queued Interaction Layer

This chapter explains the QIL (Queued Interaction Layer) service, represented by the `IQILService` interface that gets queued information from Genesys servers through GIS.

This chapter includes the following sections:



## QIL Prerequisites

To follow the discussion in this chapter, you will need the *Open Media Interaction SDK 7.6 Services API Reference* (which is located in the `doc/` subdirectory of your Open Media Interaction Services SDK installation directory).

Discussion in this chapter also assumes that you read Chapter 2, "Connection," on page 21, Chapter 3, "Data Transfer Object," on page 33, and Chapter 4, "Events," on page 39 before starting reading this section.

## More QIL Essentials

The QIL (Queued Interaction Layer) service lets your applications monitor queues and get information about queued interactions in the Genesys Framework.

The `IQILService` interface provides queued data through the attribute domains and methods listed in Table 8 on page 62.

**Table 8: Domains and Related Methods**

| Domain | IQILService Methods | Events |
|---|---|---|
| `queue` | `getQueuesDTO()`<br>`startQueueMonitoring()`<br>`stopQueueMonitoring()` | `QueueEvent` |
| `qil-interaction` | | `QueueEvent`<br>`InteractionEvent` |

There are four main things you can do with the QIL service:

- Get queue data.
- Start and stop monitoring a queue.
- Monitor queues.
- Monitor queued interactions.

**Note:** You can get information about queued interactions only through events. (See the "QILInteractionDTO" topic in the *API Reference*.)

The following sections detail how to make these steps stand out so that you can quickly learn to write your own real-world applications.

# Getting Queue Data

The `queue` domain identifies data you can get for queues through the `IQILService` interface.

To get queue data, you can call the `IQILService.getQueuesDTO()` method, which returns an array of `QueueDTO` objects. This array contains a `QueueDTO` instance for each specified queue.

If you set the queueIds parameter to null when calling this method, it returns all the queues available through the `IQILService` interface.

The following code snippet shows how to make this call:

```
//Getting all queues with all queue attributes
QueueDTO[] allqueues = qilservice.getQueuesDTO(null,new String[]{"*"});
```

Within the `QueueDTO.queueId` strings, you are able to manage monitoring on queues. See the "Monitoring Queues" section immediately below.

# Monitoring Queues

Queue monitoring depends on two statuses, available in the queue data:

- `queue:status`—Indicates whether or not the queue is active. The queue is active if a strategy is loaded in the queue.
- `queue:isMonitored`—Indicates whether your application is monitoring the queue. If the queue is being monitored, you can track content changes and interaction activity.

The following subsections introduce how to use these statuses to manage queue monitoring and get events.

## Starting and Stopping Monitoring

To monitor a queue, your application must verify that this queue's status is `ACTIVE`. This is possible by getting a `QueueDTO` containing the `queue:status` attribute value for this queue.

Then, if the queue status is `ACTIVE`, your application must start monitoring this queue—for instance, `queue0`—by calling the `IQILService.startQueueMonitoring()` method, as shown in this code snippet:

```
//Getting the status of queue0
QueueDTO[] queues =
   qilservice.getQueuesDTO(new string[]{"queue0"},
      new string[]{"queue:queueId","queue:status"});
foreach(KeyValue pair in queues[0].data)
{
   if(pair.key == "queue:status")
   {
      QueueStatus status = (QueueStatus) pair.value;
      if(status == QueueStatus.ACTIVE)
      {
         qilservice.startQueueMonitoring("queue0");
      }
   }
}
```

If the request is successful, your application can get a `QueueEvent` event propagating the `MONITORED` queue monitoring status. If the queue monitoring status is `MONITORED`, your application can track queue content changes and interaction events, as detailed in the following sections.

It is as simple to stop monitoring a queue as it is to start monitoring it. Call the the `IQILService.stopQueueMonitoring()` method, as shown in this code snippet:

```
qilservice.stopQueueMonitoring("queue0");
```

# Managing Queue Events

When your application subscribes to `QueueEvent` events for the `QILService`, it can get two types of `QueueEvent` events, which propagate different attributes:

- Monitoring status changed—This event occurs when you start and stop monitoring a queue, or when a queue modification could have changed the monitoring status.
- Queue content changed—This event occurs on a `MONITORED` queue when the queue content changed, that is, when interactions were added to or removed from the queue.

Table 9 shows which queue attributes are published with each event.

### Table 9: QueueEvents and Attributes

| QueueEvent Type | Attribute to test | Associated Attributes |
|---|---|---|
| Monitoring status changed | `queue:queueChanged` | `queue:queueChangeReason`<br>`queue:newMonitorStatus`<br>`queue:oldMonitorStatus` |
| Queue content changed | `queue:queueContentChanged` | `queue:queueAddedInteractions`<br>`queue:queueRemovedInteractions` |

To monitor a queue, you must track the `QueueEvent` events for changes in monitoring status. You can get content changes and interaction events on a queue only if this queue's monitoring status is `MONITORED`.

The following code snippet shows how to subscribe to `QueueEvent` events, and how to specify default attributes for getting both monitoring status changes and content changes on two queues—`queue0` and `queue1`.

```
/// 1. Creating a TopicsService for the QIL Service
TopicsService queueTopicsService = new TopicsService();
queueTopicsService.serviceName = "QILService";
queueTopicsService.topicsEvents = new TopicsEvent[1] ;

/// 2. Defining a topic event for QueueEvent
queueTopicsService.topicsEvents[0] = new TopicsEvent() ;
queueTopicsService.topicsEvents[0].eventName = "QueueEvent" ;

//Setting attributes to retrieve
queueTopicsService.topicsEvents[0].attributes = new string[]{"default"};

/// 3. Adding triggers
queueTopicsService.topicsEvents[0].triggers = new Topic[2];
queueTopicsService.topicsEvents[0].triggers[0] = new Topic();
queueTopicsService.topicsEvents[0].triggers[0].key = "QUEUE";
queueTopicsService.topicsEvents[0].triggers[0].value = "queue0";
```

```
queueTopicsService.topicsEvents[0].triggers[1] = new Topic();
queueTopicsService.topicsEvents[0].triggers[1].key = "QUEUE";
queueTopicsService.topicsEvents[0].triggers[1].value = "queue1";

///4. Subscribing with the subscriber ID of the application
TopicsService[] newTopics = new TopicsService[]{queueTopicsService};
eventservice.subscribeTopics( mySubscriber.subscriberId,newTopics);
```

## Managing Interaction Events

If a queue's monitoring status is `MONITORED`, you can get events for interactions of this queue, that is, `InteractionEvent` events. These events involve status and property changes in interactions.

These events let your application access the `QILInteractionDTO` objects that contain detailed information about an interaction, described in the `qil-interaction` domain of the `IQILService`. For further details, see the *Open Media Interaction SDK 7.6 Services API Reference*.

The following code snippet adds a new topic service for the QIL service, which lets the subcribing application receive `InteractionEvent` events, which occur on interactions of `queue0`.

```
/// 1. Creating a new TopicsService for the QIL Service
TopicsService newqueueTopicsService = new TopicsService();
newqueueTopicsService.serviceName = "QILService";
newqueueTopicsService.topicsEvents = new TopicsEvent[1] ;

/// 2. Defining a topic event for InteractionEvent
newqueueTopicsService.topicsEvents[0] = new TopicsEvent() ;
newqueueTopicsService.topicsEvents[0].eventName = "InteractionEvent" ;

//Setting attributes to retrieve
newqueueTopicsService.topicsEvents[0].attributes = new string[]{"default"};

/// 3. Adding triggers
newqueueTopicsService.topicsEvents[0].triggers = new Topic[1];
newqueueTopicsService.topicsEvents[0].triggers[0] = new Topic();
newqueueTopicsService.topicsEvents[0].triggers[0].key = "QUEUE";
newqueueTopicsService.topicsEvents[0].triggers[0].value = "queue0";


///4. Subscribing with the subscriber ID of the application
TopicsService[] newTopics = new TopicsService[]{newqueueTopicsService};
eventservice.subscribeTopics( mySubscriber.subscriberId,newTopics);
```

Chapter

# 7 Media Interaction Layer

This chapter explains the MIL (Media Interaction Layer) service, represented by the `IMILService` interface. This service manages third-party media interactions in Interaction Server through GIS.

This chapter also explains the UCS (Universal Contact Server) service, represented by the `IUCSService` interface, which manages third-party media interactions' data in UCS through GIS.

This chapter includes the following sections:



## Prerequisites

To follow the discussion in this chapter, you will need the *Open Media Interaction SDK 7.6 Services API Reference* (which is located in the `doc/` subdirectory of your Open Media Interaction Services installation directory).

Discussion in this chapter also assumes that you have read Chapter 2, "Connection," on page 21, Chapter 3, "Data Transfer Object," on page 33, and Chapter 4, "Events," on page 39.

You may need to read to learn more about the business attributes that characterize third-party media interactions.

# More MIL Essentials

The MIL (Media Interaction Layer) service and the UCS service include all methods required to manage third-party media interactions (also called MIL interactions). This section discusses the main features available through the API.

All interfaces and atomic methods provided with the MIL and UCS services for handling interactions allow you to build your own interaction workflow and manage interaction-related information (status, attached data, and so on).

Therefore, when developing your applications, pay attention to synchronizing interaction data through servers, such as Interaction Server and UCS.

## MIL Service

The `IMILInteractionService` interface provides synchronous methods to perform requests like the following on MIL interactions handled by Interaction Server:

- Submit an interaction with its interaction parameters.
- Stop processing interactions.
- Change submitted interactions' parameters.
- Get callback requests on an interaction you created.

**Note:** You cannot use the MIL service to manage Genesys multimedia interactions, that is, interactions of type `chat` or `email`.

Your application cannot get events on MIL interactions' changes (status, data, and so on.) It can subscribe to `CallbackEvent` events, provided with ESP (External Service Protocol) features.

ESP is a Genesys protocol that the MIL service can use on the GIS side to communicate with Interaction Server. Its purpose is to let Interaction Server send requests to your application depending on external services defined in your routing strategies.

For further details about these interfaces, see the *Open Media Interaction SDK 7.6 Services API Reference*. For details about implementing external services, refer to your MCR 7.6 and Universal Routing 7.6 documentation. For details about ESP, see "Managing ESP Callbacks" on page 71 of this guide.

## UCS Service

The `IUCSService` interface provides UCS features that enable your application to:

- Save MIL interactions before or after their submission.

- Get, or search for, MIL interactions saved in the UCS database.
- Stop processing, or delete, MIL interactions saved in the UCS database.

When saving a MIL interaction, you should pay attention to its UCS parameters—see the `mil-interaction.ucs` domain. For logical reasons, some parameters are set only once. An example is the `mil-interaction.ucs:CanBeParent`, which specifies whether or not the saved interaction can be a parent interaction of other interactions saved in UCS. For further details about these interfaces and domains, see the *Open Media Interaction SDK 7.6 Services API Reference*.

Genesys recommends that when your application modifies MIL interactions' data through the `IMILService`, your application should propagate these modifications in the UCS database using the `IUCSService` interface.

# Submitting a MIL Interaction

The MIL (Media Interaction Layer) service includes all the required methods to manage the creation of new MIL interactions, then submit them to the Genesys Framework and servers. This is, for instance, to submit inbound third-party media interactions to the Genesys environment.

When submitting a new interaction, your application must specify its main characteristics, including an interaction ID, as shown in the following code snippet.

```
KeyValue[] interactionData = new KeyValue[1];
interactionData[0] = new KeyValue();
interactionData[0].key = "mil-interaction:receivedAt";
//long date for 03/17/2002 15:58:41
long receivedAt = 126608503795055696;
interactionData[0].value = receivedAt;

MILInteractionDTO result =
   milservice.submitInteraction("interaction0", "queue0", "Inbound","New",
    "ThirdPartyMedia", interactionData, false, new String[]{"mil-interaction:*"});
```

The provided interaction ID must be unique, otherwise Interaction Server will throw an exception and the submission will fail. If you set the parameter for saving the submitted interaction in UCS to `true`, the interaction ID must be less than or equal to 16 characters long.

## Managing Interaction Data

If your application calls the `IMILService.submitInteraction()` method, the interaction data parameter can include two types of interaction parameters, identified in two distinct domains of the MIL service:

- `mil-interaction.is`—Parameters for Interaction Server. They are always submitted with the interaction to Interaction Server.
- `mil-interaction.ucs`—Parameters for UCS. If the `save` parameter of the `IMILService.submitInteraction()` method is set to `true`, UCS parameters are saved in UCS with the interaction.

The `mil-interaction:ucs` parameters are not submitted to Interaction Server. If you need both to save and submit some properties, such as attached data, you must add them both as `mil-interaction.is` and as `mil-interaction.ucs` data.

If your application changes interaction properties by calling the `IMILService.changeProperties()` method, this call modifies only the properties corresponding to the `mil-interaction.is:properties` attribute. It does not modify the `mil-interaction.ucs:properties` properties. If you wish to make these `mil-interaction.ucs:properties` properties persistent in UCS, modify this attribute and save the interaction.

# Managing Interactions in UCS

This section provides details about MIL interaction management in UCS.

## Getting Interaction Data from UCS

When developing your application, you should take into account the cache mecanism that lets your application get interaction information with the UCS service by calling the `IUCSService.getInteractionsDTO()` method.

When requesting interaction information from UCS, the `IUCSService` service retrieves part of the interaction data, that is, the information being processed in a `MILInteractionDTO` object.

This information corresponds to the values of the following attributes: `mil-interaction.ucs:properties`, `mil-interaction.ucs:contentBinary`, `mil-interaction.ucs:contentBinarySize`, and `mil-interaction.ucs:contentMimeType`.

Then, even if you modify this interaction by setting new values for writable attributes, modifications are not available upon the next call to the `IUCSService.getInteractionsDTO()` method.

For example, when your application reads the UCS status of this interaction, that status remains `UCSInteractionStatus.UNKNOWN`. If you assign a new UCS status to the interaction, for instance, `UCSInteractionStatus.PENDING`, this status is updated in UCS when your application saves the interaction. But, the

next time your application gets a `MILInteractionDTO` object for this interaction, the UCS status will be `UCSInteractionStatus.UNKNOWN`.

## Saving MIL Interactions in UCS

When saving a `MILInteraction` object, you should pay attention to its `mil-interaction.ucs:*` attributes. For logical reasons, some of these attributes are set only once. An example is `mil-interaction.ucs:canBeParent`, which specifies whether or not the interaction can be a parent interaction of other interactions saved in UCS.

For further details about these interfaces, see the *Open Media Interaction SDK 7.6 Services API Reference*.

# Managing ESP Callbacks

ESP callbacks are `CallbackEvent` events of the `IMILService`. You can subscribe to these events, so that when Interaction Server sends a request to your application through GIS for a particular MIL interaction, your application gets a `CallbackEvent`.

In this event, you can get `mil-callback:*` information, including a reference ID for the callback request. Within this information, your application can handle this request. Then, using the `MILService` methods, you can send an error or success response to Interaction Server.

To properly handle ESP callbacks, you need to perform the steps detailed in the following subsections:

- "Defining an ESP Strategy" on page 71
- "Subscribing to Callback Events" on page 72
- "Managing ESP requests" on page 73

## Defining an ESP Strategy

A callback request is always associated with an ESP strategy and identified as an External Service. You can create several types of callback requests and each of them must correspond to an external service.

To define an ESP strategy, start the Interaction Routing Designer and create an External Service. In the `External Service Property` box, specify the fully qualified name of the service (for example, `com.genesyslab.examples.mil.ServiceName`) and the `methodName` method to call during the script execution.

In this example, the trigger identifying the callback type in the Open Media Services would be:

```
NAME=com.genesyslab.examples.mil.ServiceName.methodName
```

You can make use of this callback type to define a `ServiceName` class handling this callback type, as for example below:

```
using com.genesyslab.examples.mil;
public class ServiceName
{
   public void methodName(String refId, String milId, KeyValue[]
parameters,KeyValue[] data)
   {
      //code handling the callback request
      //...
   }
}
```

Refer to your MCR 7.6 and Universal Routing 7.6 documentation for details about implementing external services.

## Subscribing to Callback Events

To subscribe to callback events, create `TopicsEvents` for each callback type associated with an external servive. In the following code snippet, the application subscribes to a callback type defined in “Defining an ESP Strategy” on page 71.

```
/// Creating a new TopicsService for the MIL Service
TopicsService milTopicsService = new TopicsService();
milTopicsService.serviceName = "MILService";
milTopicsService.topicsEvents = new TopicsEvent[1] ;

/// Defining a topic event for CallbackEvent
milTopicsService.topicsEvents[0] = new TopicsEvent() ;
milTopicsService.topicsEvents[0].eventName = "CallbackEvent" ;

//Setting attributes to retrieve
milTopicsService.topicsEvents[0].attributes =
   new string[]{"default"};

/// Adding trigger for the ServiceName callback
milTopicsService.topicsEvents[0].triggers = new Topic[1];
milTopicsService.topicsEvents[0].triggers[0] = new Topic();
milTopicsService.topicsEvents[0].triggers[0].key = "NAME ";
milTopicsService.topicsEvents[0].triggers[0].value =
   "com.genesyslab.examples.mil.ServiceName.methodName";

/// Subscribing with the subscriber ID of the application
TopicsService[] newTopics = new TopicsService[]{milTopicsService};
eventservice.subscribeTopics( mySubscriber.subscriberId,newTopics);
```

# Managing ESP requests

If your application gets a `CallbackEvent` event through the event service, it is supposed to handle the associated callback request. To handle this request, you add some source code to perform the request, then to send a response to Interaction Server by calling a method of the MIL service:

- `IMILService.sendFault()`—Sends a fault response to Interaction Server.
- `IMILService.sendResponse`—Sends a successful response, including data, to Interaction Server.

For example, in the following code snippet, the application creates an instance of the `ServiceName` class and call its `methodName` method to handle the callback request, then sends a successful response.

```
Event[] events =
   eventservice.getEvents(mySubscriber.subscriberId,1);
foreach(Event evt in events)
{
   if(evt.serviceName=="MILService" &&
        evt.eventName =="CallbackEvent")
   {
      //There is a single trigger for this application,
      //so we do not check triggers in this example

      String referenceID = null;
      string milInteractionID = null;
      KeyValue[] parameters = null;
      KeyValue[] userData = null;

      foreach(KeyValue pair in evt.attributes)
      {
         if(pair.key =="mil-callback:referenceId")
         {
            referenceID = (string) pair.value;
         }
         else if(pair.key =="mil-callback:interactionId")
         {
            ///...
         }
         //...
      }
      // Managing the request
      ServiceName srv = new ServiceName();
      srv.methodName(referenceID, milInteractionID,
         parameters,userData);

      //Send a response to Interaction Server
      milservice.sendResponse(referenceID,null,null);
   }
   //...
```

```
}
```


# Index

## K



## M



## N



## P



## Q



## S



## T



## U



## V



## W



## X