Agent Interaction SDK Services Developer Guide

About Proxies and Examples

12/13/2025

Contents

About Proxies and Examples

This release of the documentation includes code snippets in most chapters. These code snippets illustrate many product features and can serve as a basis for developing your own applications.

> **Important**
> To download the code samples, see the Agent Interaction SDK resources.

The code snippets in this *Developer's Guide* are in C#, but there are differences across these examples depending on the generated proxy and the language. For instance, in the *Agent 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 *Agent Interaction SDK 7.6 Services API Reference for the Java Proxy,* on the other hand, service interfaces are defined in accordance with the following rule: `<service_name>Service`.

> **Important**
> To download the API references, see the Agent Interaction SDK resources.

You can do any of the following:

- Use a toolkit to generate a proxy from the provided WSDL files.
- Use one of the provided .NET proxies available on the product CD in the `tools/` directory.
- Use one of the provided Java proxies available on the product CD in the `tools/` directory.

Then, use the chosen proxy to build your desktop application using the Agent Interaction SDK Service that the Agent Interaction Services Libraries expose.

Generating a SOAP Proxy

You can use a toolkit to generate a SOAP proxy from the provided WSDL files—for example, Apache AXIS toolkit, version 1.1 or 1.3, for Java development (for further information, see: `http://ws.apache.org/axis/java/user-guide.html`).
With a SOAP proxy, use the GIS session service to connect your client application and set options. Refer to the *Statistics SDK 7.6 Web Services Developer's Guide* for further details about the session service, and see this chapter's API Overview for further details about available options.

Opening a Session

The first step your Agent Interaction SDK Services client application must perform is to open a session in GIS to get a session ID, which must be passed in the URL of all 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 an agent service using a stub created with /// Apache Axis toolkit 1.1

import com.genesyslab.www.ail.*;
import com.genesyslab.www.ail.agent.*;

//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);

// Accessing Services
String[] value = sessionService.getServices(new java.lang.String[] {
"GIS_INTERACTIONSERVICE"});

AgentServiceSoapBindingStub agentService =
   (AgentServiceSoapBindingStub) new AgentService_ServiceLocator().getAgentService();

agentService.setTimeout(60000);

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

// then using agent service is similar to C#
// logging an agent

LoginVoiceForm loginVoiceForm = new LoginVoiceForm();
loginVoiceForm.setLoginId(loginId);
loginVoiceForm.setWorkmode(WorkmodeType.AFTERCALLWORK);
MediaInfoError[] values = agentService.login(agentId, placeId, loginVoiceForm, null);
```

Using the .NET Proxy

You can use the provided .NET proxy to minimize session management tasks and to simplify service creation. This proxy is available in the tools/ directory on the GIS Product CD.

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

Service Factory

The `com.genesyslab.ail.ServiceFactory` class is the entry point for the .NET proxy. You must create a `ServiceFactory` object in order to connect. The connection can be synchronous 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 until a connection succeeds or the factory is released. To monitor the connection status, you must specify an `IServiceFactoryListener` listener at 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 `ServiceFactory` creation. See XML Configuration File for .NET for details about options.
- You can use an XML file to configure your `ServiceFactory` object.

Using an XML file is simple: write your own XML file that defines the factory parameters, or use the default `ail-configuration.xml file; then, indicate the factory parameters to be used. The following code snippet shows the ServiceFactory creation based on the WebServicesFactory factory defined in the ail-configuration.xml file.`

```
// Instantiation of a ServiceFactory to make the connection
ServiceFactory myServiceFactory =
   ServiceFactory.createServiceFactory("WebServicesFactory", null, null);
```

See XML Configuration File for .NET for further details.

> **Important**
>
> An `ail-configuration.xml` file is available on the GIS product CD in the `tools/` directory.

Access Services

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 iservice =
myServiceFactory.createService(typeof(IXxxService), null) as IXxxService;
```

If the `Hashtable` parameter is `null` in the `createService()` call, the Agent Interaction Service layer takes into account the current context of the factory. Otherwise, the Agent Interaction Service layer

uses the specified `Hashtable` for service creation.
The following code snippet shows how to create an agent service:

```
IAgentService myAgentService = myServiceFactory.createService(typeof(IAgentService), null) as
IAgentService;
```

> **Important**
> Your application can typically use the current factory context for service creation.

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 SOAP:
  - `WebServicesFactory` —The factory name.
  - `url` option—The value is `http://[Server Address]:[Server Port]/gis`.

The following sections present the optional attributes, based on proxy type, attached to the mandatory attributes specified above.

XML Optional Attributes

The following table shows all the attributes that you can define.

Optional 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]`
`Timeout`	int	The timeout interval for an XML web

Name	Type	Description
		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.
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.

Name	Type	Description
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="WebServicesFactory" classname="com.genesyslab.ail.WebServicesFactory"
assembly="AilLibrary">
<option name="Url" value="http://[Server Address]:[Server Port]/gis" />
<option name="gis.username" value="default" />
<option name="gis.password" value="password" />
< !—- OPTIONAL
<option name="gis.sessionId" value="1234567"/>
<option name="notification.HTTPport" type="int" value="10000"/>
<option name="notification.createHTTPchannel" type="bool" value="true"/>
<option name="notification.objectURI" value="NotifLoad"/>
<option name="gis.checkSessionInterval" type="int" value="900"/>
<option name="service-point-manager.defaultConnectionLimit" type="int" value="10"/>
<option name="notification.reachableURL" value="http://localhost:8080/ail"/>
<option name="service-point-manager.maxServicePointIdleTime" type="int" value="90000"/>
END OPTIONAL — >
</factory>
</configuration>
```

HTTP Redirections

By default, redirections are disabled at startup. To enable redirections, you must set the AllowAutoRedirect property to true as follows:

```
WebServicesFactory wsf = mAilServiceFactory.ServiceFactoryImpl as WebServicesFactory;
```

```
wsf.gisSessionService.AllowAutoRedirect = true;
```

This option is dynamic and can be modified at runtime or during the compilation.
The following table lists the supported HTTP codes in this configuration.

Redirection Codes tested with the .NET Proxy

HTTP Code	Supported	Description
300	Yes	POST, then GET to the new URL.
301	Yes	POST, then GET to the new URL.
302	Yes	POST, then GET to the new URL.
303	Yes	POST, then GET to the new URL.
304	Yes	No redirect.
307	Yes	POST, then POST to the new location.
308	No	No redirect.

Using the Java Proxy

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.
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 for the default configuration file name, the process and the method to be called are identical to those described in Service Factory.

> **Important**
>
> The default XML configuration filename is `proxy-configuration.xml`. For further details, see XML Configuration File for Java.

Access Services

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

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]/gis`.

Your application reads the XML configuration file—by default, `proxy-configuration.xml` —to determine which protocols and options should be used for instanciating its connection to GIS. The following sections present the optional attributes, according to proxy type, attached to the mandatory attributes specified above.

XML Optional Attributes

The following table shows all the attributes that you can define.

Optional Attributes

Name	Description
`Username`	Username pour basic authentification.
`Password`	Password pour basic authentification.
`backupUrls`	List of backup connection urls to be used in case of disconnection.

Name	Description
`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.
`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.

Name	Description
`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 Agent Interaction Services 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"/>
<option name="DocumentMode" value="false"/>
<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>
```

GIS License

SOAP

The `GIS_INTERACTIONSERVICE` license is checked out when your application needs to call the `ServiceFactory.createFactory()` method. To check the license in, your application calls the `ServiceFactory.releaseFactory()` method.
In a scenario where the agent logs out and the application properly terminates, your application should release the ServiceFactory instance when the application ends; this frees the agent's license. In an application crash scenario, the license is checked in when the GIS session ends, as is also the case with the `GIS_CONFIGURATION_SERVICE` and `GIS_STATSERVICE` licenses.

HTTP Redirections

Unlike with the .NET proxy, you cannot enable or disable redirections at compilation or runtime. In Java, the Axis client handles the HTTP connection through a library, which is defined in the `wsdd` client file, and relies on the v3.0.1 Jakarta HTTP client (see HttpClient Home.)
For instance, to enable redirections, you should disable the `HttpCommonsSender` library in the Java client that implements the proxy library:

```
context.setProperty("EnableHttpCommonsSender", "false");
```

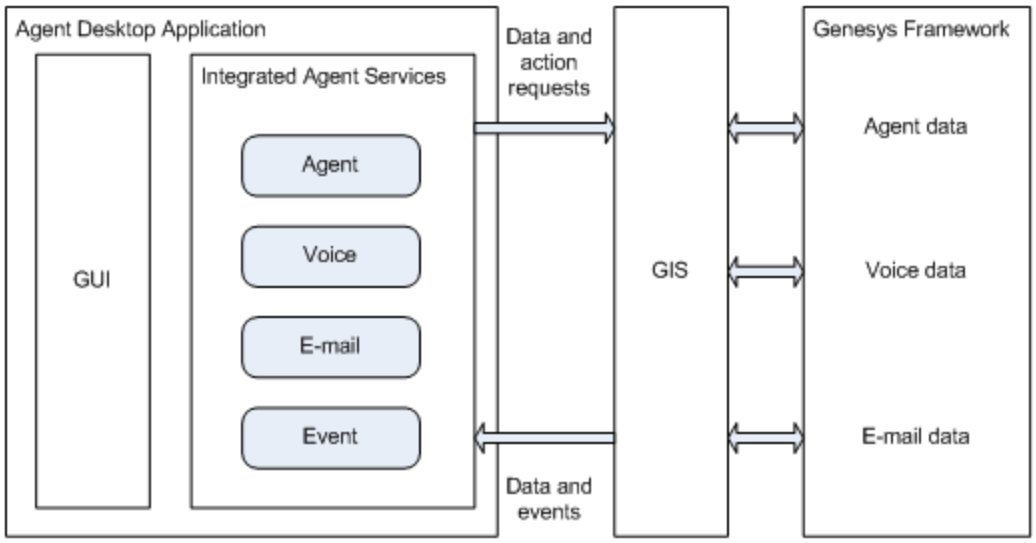
Then, you can provide your own `client-config.wsdd` file which defines the transport layer, as shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<deployment name="defaultClientConfig"
xmlns="http://xml.apache.org/axis/wsdd/"
xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
<globalConfiguration>
<parameter name="disablePrettyXML" value="true"/>
<parameter name="enableNamespacePrefixOptimization" value="false"/>
<parameter name="sendMultiRefs" value="false"/>
</globalConfiguration>
<transport name="http" pivot="java:com.genesyslab.soa.impl.channel.axis.handler.HTTPSender" />
<transport name="https" pivot="java:org.apache.axis.transport.http.CommonsHTTPSender"/>
<transport name="local" pivot="java:org.apache.axis.transport.local.LocalSender"/>
</deployment>
```

To develop a transport layer, you should extend the `org.apache.axis.handlers.BasicHandler` class. Refer to the official Axis documentation for further information.

API Overview

The Agent Interaction Services API works with the mirroring services available in GIS. Typically, your application is a client application, integrating Agent Interaction Services to perform agent actions and to communicate data and events with the Genesys Framework, as shown in the figure below.



Services Integrated in an Agent Application

Your application deals with services that transparently hide GIS and the Genesys Framework, which handles information and CTI objects. These services provide the following features:

- handling agent activity, such as login and logout.
- handling voice interactions such as answering, calling, and callback.
- handling e-mail interactions.
- handling outbound campaigns.
- using the Standard Response Library.
- handling contacts and their histories.

One particularity of the Agent Interaction Services API is its *service approach design* This means that a service deals with dedicated information concerning a set of remote objects. For example, the agent service deals with agent data only.

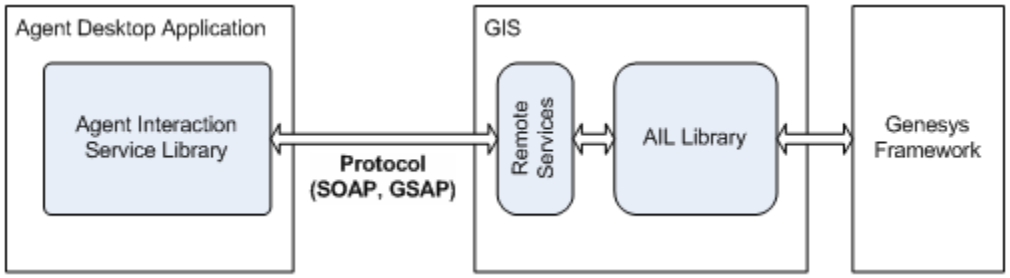
Building an Application Using Services

Take the following steps (a general strategy) to build your application on the Agent Interaction Services API:
Connect your application. See Opening a Session.

1. Get an event service. The Event Service.
2. Get an agent service. The Agent Service.
3. Subscribe to events.
4. Get the services required to implement your application features.
5. Update your application any time you retrieve an event according to possible actions or status changes.

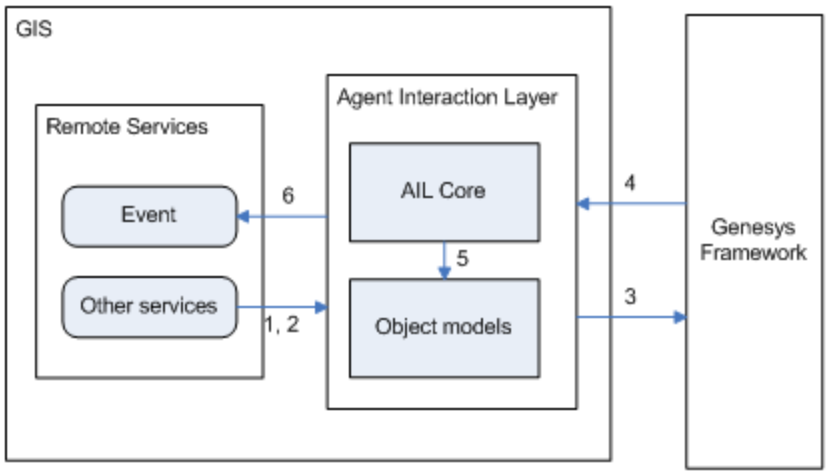
The Remote Services

When using the Agent Interaction Services API, you are dealing with remote services integrating the Agent Interaction Layer library. This library is part of the Genesys Agent Interaction (Java API). See the Agent Interaction SDK Java documentation for full details of AIL features.
The AIL library internally implements models of Genesys products, such as Framework voice calls, e-mail, outbound campaigns, and so on.
The remote services are exposed in GIS, as shown in the following figure.



The Remote Services and the Genesys Interface Server

On the GIS side, the remote services implement the AIL library, which internally handles models of the Genesys Framework. On the agent-desktop-application side, the integrated services hide the interactions with the remote services which encapsulate the AIL library.
The AIL library maintains models of the Genesys objects used by your application. These objects might include, for example, DNs, a `Places`, agents, or interactions. The native AIL library API offers interfaces to perform actions on these objects, and the library core internally handles the state models, as illustrated in the following figure.



The Remote Services and the AIL Internal Core

The numbered labels in the above figure describe the following actions:

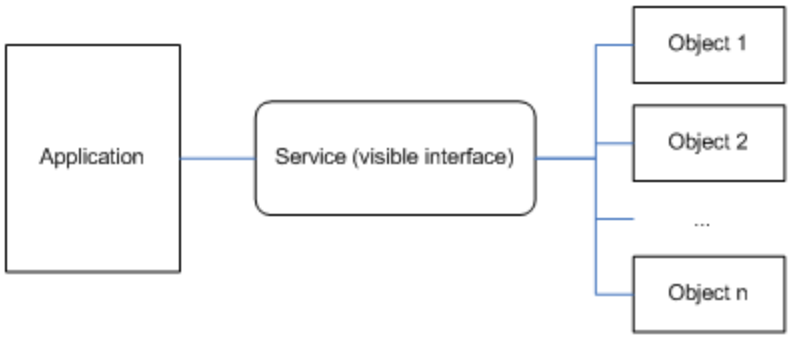
1. Implementation of an AIL Interface dealing with a core object
2. Calling an AIL object method
3. AIL Sending a set of requests
4. AIL Receiving Events
5. AIL core updating core object models
6. AIL sending events to the Event Service

As shown in the above figure, the AIL core notifies the remote services with events when the states of objects or data change. For example, if your service requests an agent login on a media type, once the agent is logged in, the AIL core notifies the event service that the agent status on the media is now `READY` or `NOT_READY`. If your application has registered to listen to events on the agent, it receives the event and can inspect the data.

> **Important**
>
> Genesys recommends that you base your agent desktop application on the state models provided by the Agent Interaction Services API.

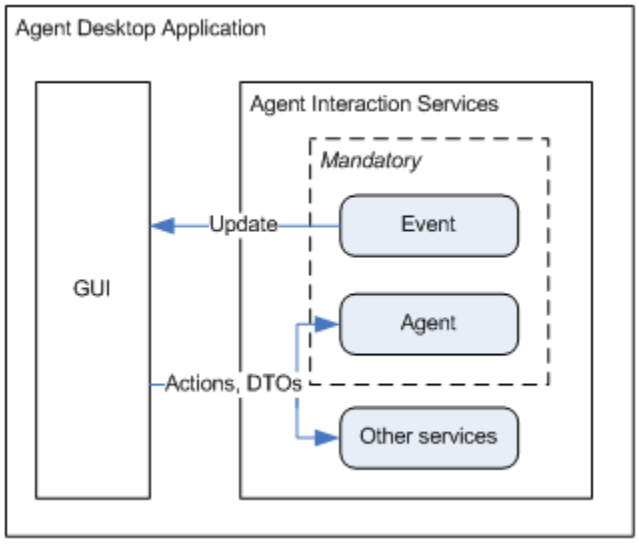
Using the Services

The Agent Interaction Services API provides a set of services. A service is an interface dealing with a group of objects. For example, an interaction service gives access to interactions' data. A single request can apply to a group of several interactions.



Service Concept

Services are agent-oriented; that is, services are designed to fulfill requests of agent type and should be used for this purpose.
The Agent Interaction Services API represents services suitable to perform agent actions and to retrieve any data required by an agent-oriented application. How you make use of the various services is a matter of your application design.
Your application integrates at least two services: the agent service and the event service, as shown in the following figure.



Using the Services in Your Agent Desktop Application

To properly use other services, your application requires:

- An agent service—This service manages agent actions and data; without an agent logged in with this service, some services cannot fulfill actions and data requests.
- An event service—Your application must update according to the data propagated in events which are

published in this service.

Possible Actions

The Agent Interaction Service API is designed to implement agent desktop applications. Therefore, services have been designed to perform agent actions that affect the components managed by the Genesys Framework.
For example, getting ready, logging in, and logging out on media are actions handled by the `IAgentService` interface, and performed by calls to the corresponding methods.
However, a given service's actions might not all be available at a given point in time. An obvious example is an agent not being allowed to become `READY` on media where he or she has not yet logged in.
Thus, to perform a particular action, your application must first check to see if this action is possible:

- Actions are identified in the `<service_name>Action` enumeration.
- Possible actions are provided with attributes of the `I<service_name>Service` interface attributes.
- Possible actions are updated and propagated with events.

A good use of possible actions is enabling or disabling graphical components that the agent uses to perform actions.

Statuses

The services interact with objects and provide you with their statuses during runtime.

- Statuses are identified in the `*Status` enumeration of service namespaces.
- Status access is provided with attributes of the `I<service_name>Service` interface.
- Status changes are propagated in events.

You should base your application design on these statuses. For example, in a GUI context, if a call status is `IDLE` (terminated), the associated GUI components have no need to be visible anymore.

> **Important**
> To determine which actions are available in the current status of the application, rely on the provided possible actions.