Platform SDK Developer's Guide

Universal Contact Server

12/14/2025

Contents

Universal Contact Server

You can use the Contacts Platform SDK to write Java or .NET applications that interact with the Genesys Universal Contact Server (UCS). This allows you to create applications that work with contacts, interactions, and standard responses in a variety of ways - either to create a full-featured agent desktop, or a simple application that forwards email messages.

This document shows how to implement the basic functions you will need to write simple UCS-based applications.

When you are ready to write more complicated applications, take a look at the classes and methods described in the Platform SDK API Reference.

Java

Using the Contacts Protocols

Before using the Contacts Platform SDK, you should include `import` statements that allow access to the Platform SDK Commons and Contacts classes:

```
[Java]

import com.genesyslab.platform.commons.protocol.*;

import com.genesyslab.platform.contacts.protocol.*;
import com.genesyslab.platform.contacts.protocol.contactserver.*;
import com.genesyslab.platform.contacts.protocol.contactserver.events.*;
import com.genesyslab.platform.contacts.protocol.contactserver.requests.*;
```

Setting Up Universal Contact Server Protocol Objects

The first thing you need to do to use the Contacts Platform SDK is instantiate a `UniversalContactServerProtocol` object. To do that, you must supply information about the Universal Contact Server you want to connect with. This example uses the server's name, host, and port information, but you can also use just the URI of your Universal Contact Server:

```
[Java]

UniversalContactServerProtocol ucsConnection = new UniversalContactServerProtocol(new
Endpoint(universalContactServerURI));
```

It is a good practice to always set the application name at the same time that you instantiate a new protocol object. This application name will be used to identify where UCS requests came from.

This is also a good time to add event handlers to the protocol object. See the Event Handling section in this introductory material for code samples and details.

```
[Java]

// Set the ApplicationName property
ucsConnection.setClientName("IntroducingContactsPSDK");
```

After setting up your protocol object, the code to open a connection to the server is simple:

```
[Java]

ucsConnection.open();
```

> **Tip**
>
> Be sure to use proper error handling techniques in your code, especially when working with the protocol connection. To save space, these error handling steps are not shown in this example.

Inserting an Interaction

Now that the protocol connection is open, you are ready to start handling interactions. In this example, we will start by creating a new, outbound email interaction using the `RequestInsertInteraction` request.

Creating a new email interaction object takes a bit of planning. Before you can create and submit the request object, you need to create and configure the following objects:

- `InteractionAttributes` - Sets common attributes for this interaction, specifying details such as the media type and status. All interactions need these attributes to be configured.
- `EmailOutEntityAttributes` - Sets attributes that are specific to an outbound email interaction. For outbound email interactions, this includes the sending and receiving addresses. (The type of interaction you are creating will dictate which object to use here; for example, phone interactions require a `PhoneCallEntityAttributes` object instead of `EmailOutEntityAttributes`.)
- `InteractionContent` - Specifies the actual interaction content. This can be Text, MIME, StructuredText, or StructuredText with MIME content.

The following code snippet shows how each of these objects is configured for our simple outbound email example:

```
[Java]

// Set common interaction attributes
InteractionAttributes attributes = new InteractionAttributes();
attributes.setTenantId(101);
attributes.setMediaTypeId("email");
attributes.setTypeId("Outbound");
attributes.setSubtypeId("OutboundRedirect");
attributes.setStatus(Statuses.Pending);
```

```
attributes.setSubject(subjectLine);
attributes.setQueueName(queueName);
attributes.setEntityTypeId(EntityTypes.EmailOut);

// Set entity-specific attributes
EmailOutEntityAttributes outEntityAttributes = new EmailOutEntityAttributes();
outEntityAttributes.setFromAddress(fromAddress);
outEntityAttributes.setToAddresses(forwardAddress);

// Set interaction content
InteractionContent content = new InteractionContent();
content.setText("Email message text...");
```

> **Tip**
>
> The `InteractionAttributes` class stores the `StartDate` property in UTC format. If no value is provided, UCS uses the current date.

Once you have configured the attributes and content for the interaction, it is easy to create and submit the new request:

```
[Java]

// Create the new interaction request
RequestInsertInteraction request = RequestInsertInteraction.create();
request.setInteractionAttributes(attributes);
request.setEntityAttributes(outEntityAttributes);
request.setInteractionContent(content);

// Submit the request
EventInsertInteraction eventInsertIxn = (EventInsertInteraction)
ucsConnection.request(request);
```

Adding an Attachment

Now that you know how to create new email interactions, it is the perfect time to learn how to add attachments to existing interactions. The process for this is much easier than creating a new interaction; you just need to create the request and specify the attachment properties as shown in the code snippet below. Once the request is ready, submit it to your UCS protocol object.

```
[Java]

RequestAddDocument request = RequestAddDocument.create();
request.setInteractionId(eventInsertIxn.getInteractionId());
request.setDocumentId(strDocumentId);
request.setDescription(strDescription);
request.setMimeType(strMimeType);
request.setTheName(strName);
request.setTheSize(intSize);

EventAddDocument eventAddDocument = (EventAddDocument) ucsConnection.request(request);
```

Note that before adding an attachment, you need to have the Interaction ID available. In our

example, the Interaction ID was returned as part of the `EventInsertInteraction` from the previous section. Otherwise we would need to submit a `RequestGetInteractionContent` request and then take the Interaction ID from the resulting event.

Getting an Interaction from UCS

Now that we have created a new Interaction and submitted it to UCS, what happens next? The final task we will cover in this introduction is how to return the Interaction and any of its attachments for processing.

The structure of `RequestGetInteractionContent` is very basic: set the Interaction ID you are looking for, and then use the `IncludeAttachments` and `IncludeBinaryContent` properties to specify what type of content you want to be returned. In this example, we will return the attachment created previously and store it in an `Attachment` object for later use.

```
[Java]

RequestGetInteractionContent request = RequestGetInteractionContent.create();
request.setInteractionId(interactionId);
request.setIncludeAttachments(true);

EventGetInteractionContent eventGetIxnContent = (EventGetInteractionContent)
ucsConnection.request(request);

String subject = eventGetIxnContent.getInteractionAttributes().getSubject();
String key = eventGetIxnContent.getInteractionAttributes().getId();
if (eventGetIxnContent.getAttachments() != null) {
        Attachment attachedFile = eventGetIxnContent.getAttachments().get(0);
}
```

Closing the Connection

Finally, when you are finished communicating with the server, you should close the connection and dispose of the object to minimize resource utilization:

```
[Java]

if (ucsConnection.getState() != ChannelState.Closed && ucsConnection.getState() !=
ChannelState.Closing)
        {
                ucsConnection.close();
        }
```

.NET

Using the Contacts Protocols

Before using the Contacts Platform SDK, you should include using statements that allow access to the Platform SDK Commons and Contacts namespaces:

[C#]

```
using Genesyslab.Platform.Commons.Protocols;
using Genesyslab.Platform.Contacts.Protocols;
using Genesyslab.Platform.Contacts.Protocols.ContactServer;
using Genesyslab.Platform.Contacts.Protocols.ContactServer.Requests;
using Genesyslab.Platform.Contacts.Protocols.ContactServer.Events;
```

Setting Up Universal Contact Server Protocol Objects

The first thing you need to do to use the Contacts Platform SDK is instantiate a `UniversalContactServerProtocol` object. To do that, you must supply information about the Universal Contact Server you want to connect with. This example uses the server's name, host, and port information, but you can also use just the URI of your Universal Contact Server:

[C#]

```
UniversalContactServerProtocol ucsConnection;
ucsConnection = new UniversalContactServerProtocol(new Endpoint("UCS", ucsHost, ucsPort));
```

It is a good practice to always set the application name at the same time that you instantiate a new protocol object. This application name will be used to identify where UCS requests came from.

This is also a good time to add event handlers to the protocol object. See the Event Handling article for details.

[C#]

```
// Set the ApplicationName property
ucsConnection.ClientName = "IntroducingContactsPSDK";

// Add event handlers
ucsConnection.Opened += new EventHandler(ucsConnection_Opened);
ucsConnection.Error += new EventHandler(ucsConnection_Error);
ucsConnection.Closed += new EventHandler(ucsConnection_Closed);
```

After setting up your protocol object, the code to open a connection to the server is simple:

[C#]

```
ucsConnection.Open();
```

> ### Tip
> Be sure to use proper error handling techniques in your code, especially when working with the protocol connection. To save space, these error handling steps are not shown

> in this example.

Inserting an Interaction

Now that the protocol connection is open, you are ready to start handling interactions. In this example, we will start by creating a new, outbound email interaction using the `RequestInsertInteraction` request.

Creating a new email interaction object takes a bit of planning. Before you can create and submit the request object, you need to create and configure the following objects:

- `InteractionAttributes` - Sets common attributes for this interaction, specifying details such as the media type and status. All interactions need these attributes to be configured.
- `EmailOutEntityAttributes` - Sets attributes that are specific to an outbound email interaction. For outbound email interactions, this includes the sending and receiving addresses. (The type of interaction you are creating will dictate which object to use here; for example, phone interactions require a `PhoneCallEntityAttributes` object instead of `EmailOutEntityAttributes`.)
- `InteractionContent` - Specifies the actual interaction content. This can be Text, MIME, StructuredText, or StructuredText with MIME content.

The following code snippet shows how each of these objects is configured for our simple outbound email example:

```
[C#]

// Set common interaction attributes
InteractionAttributes attributes = new InteractionAttributes();
attributes.TenantId = 101;
attributes.MediaTypeId = "email";
attributes.TypeId = "Outbound";
attributes.SubtypeId = "OutboundRedirect";
attributes.Status = new NullableStatuses(Statuses.Pending);
attributes.Subject = subjectLine;
attributes.QueueName = queueName;
attributes.EntityTypeId = new NullableEntityTypes(EntityTypes.EmailOut);

// Set entity-specific attributes
EmailOutEntityAttributes outEntityAttributes = new EmailOutEntityAttributes();
outEntityAttributes.FromAddress = fromAddress;
outEntityAttributes.ToAddresses = forwardAddress;

// Set interaction content
InteractionContent content = new InteractionContent();
content.Text = "Email message text...";
```

> **Tip**
>
> The `InteractionAttributes` class stores the `StartDate` property in UTC format. If no

> value is provided, UCS uses the current date.

Once you have configured the attributes and content for the interaction, it is easy to create and submit the new request:

```
[C#]

// Create the new interaction request
RequestInsertInteraction request = RequestInsertInteraction.Create();
request.InteractionAttributes = attributes;
request.EntityAttributes = outEntityAttributes;
request.InteractionContent = content;

// Submit the request
EventInsertInteraction eventInsertIxn = (EventInsertInteraction)
ucsConnection.Request(request);
```

Adding an Attachment

Now that you know how to create new email interactions, it is the perfect time to learn how to add attachments to existing interactions. The process for this is much easier than creating a new interaction; you just need to create the request and specify the attachment properties as shown in the code snippet below. Once the request is ready, submit it to your UCS protocol object.

```
[C#]

RequestAddDocument request = RequestAddDocument.Create();
request.InteractionId = eventInsertIxn.InteractionId;
request.DocumentId = strDocumentId;
request.Description = strDescription;
request.MimeType = strMimeType;
request.TheName = strName;
request.TheSize = intSize;

EventAddDocument eventAddDocument = (EventAddDocument) ucsConnection.Request(request);
```

Note that before adding an attachment, you need to have the Interaction ID available. In our example, the Interaction ID was returned as part of the `EventInsertInteraction` from the previous section. Otherwise we would need to submit a `RequestGetInteractionContent` request and then take the Interaction ID from the resulting event.

Getting an Interaction from UCS

Now that we have created a new Interaction and submitted it to UCS, what happens next? The final task we will cover in this introduction is how to return the Interaction and any of its attachments for processing.

The structure of `RequestGetInteractionContent` is very basic: set the Interaction ID you are looking for, and then use the `IncludeAttachments` and `IncludeBinaryContent` properties to specify what

type of content you want to be returned. In this example, we will return the attachment created previously and store it in an `Attachment` object for later use.

```
[C#]

RequestGetInteractionContent request = RequestGetInteractionContent.Create();
request.InteractionId = eventInsertIxn.InteractionId;
request.IncludeAttachments = true;

EventGetInteractionContent eventGetIxnContent = (EventGetInteractionContent)
ucsConnection.Request(request);

String subject = eventGetIxnContent.InteractionAttributes.Subject;
String key = eventGetIxnContent.InteractionAttributes.Id;
if (eventGetIxnContent.Attachments != null)
{
        Attachment attachedFile = eventGetIxnContent.Attachments.Get(0);
}
```

Closing the Connection

Finally, when you are finished communicating with the server, you should close the connection and dispose of the object to minimize resource utilization:

```
[C#]

if (ucsConnection.State != ChannelState.Closed && ucsConnection.State != ChannelState.Closing)
        {
                ucsConnection.Close();
                ucsConnection.Dispose();
        }
```

Usage Tips

This section provides tips and recommended usage details for the Contacts SDK.

Getting Categories Efficiently

Introduced in release 8.0.0

In many cases, the `RequestGetAllCategories` message should not be used because it returns an excessive amount of attached information. Instead, this request can often be replaced by the following combination of services:

1. Call `GetRootCategories` - This returns only the root categories (without including sub-categories) with a limited amount of information attached: Name, Date, Type, Language
2. Manually filter categories - Decide your own filtering on root categories based on the attached information.
3. Call `GetCategory` using the root ID - Options can be passed to return a summary of sub-category or sub- Standard Response, making it possible to return a full tree without content.

4. Call `GetStandardResponse` - To get content, after filtering from `GetCategory` which Standard Response is interesting for your agent (again based on the category attributes).

This combination of services allows you to load the Knowledge Library in a more granular fashion. Prior to release 8.0.0, it was only possible to filter based on Language.

Historically, the `GetAllCategories` was intended for 3-tier servers such as Genesys Desktop. However, this request would cause the server to load the entire Category tree into memory in KeyValueCollection form when preparing the response to a client. In cases where a large Knowledge Library is mixed with a large number of `GetAllCategories` requests in parallel, consuming large amounts of memory.

Additional Topics

As support for the Platform SDKs continues to grow, new topics and examples that illustrate best-practice approaches to common tasks are being added to the documentation. For more information about using the Contacts Platform SDK, including functional code snippets, please read the following topics:

- Creating an Email - This article discusses how to use the Open Media and Contacts Platform SDKs in conjunction to create outgoing email messages.