Platform SDK Developer's Guide

Configuration

12/16/2025

Configuration

You can use the Configuration Platform SDK to write Java or .NET applications that access and update information from the Genesys Configuration Layer. These applications can range from the simple to the advanced.

This article shows how to implement the basic functions you will need to write a simple Configuration Layer application.

Once you have reviewed the information in this document, you should familiarize yourself with Configuration Layer Objects. Since the Configuration Platform SDK uses these objects for nearly everything it does, you will need to understand them before you start using this SDK.

> **Tip**
>
> The Platform SDK includes the Configuration Object Model (COM) Application Block, which is a high-performance component you can use to query on, and to create, update, and delete, Configuration Layer objects. Genesys recommends that you use this application block for most of the work you do with Configuration Layer objects.

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

Setting Up a ConfServerProtocol Object

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

```
ConfServerProtocol confServerProtocol =
        new ConfServerProtocol(
                new Endpoint(
                        confServerUri));
```

Configuration Server needs some additional information in order to create a successful connection. This information includes the type of client you wish to create, your client's name, and your user name and password:

```
confServerProtocol.setClientApplicationType(CfgAppType.CFGSCE.asInteger());
```

```
confServerProtocol.setClientName("default");
confServerProtocol.setUserName(userName);
confServerProtocol.setUserPassword(password);
```

After instantiating the `ConfServerProtocol` object, you need to open the connection to the Configuration Server:

```
confServerProtocol.open();
```

Creating a Query

Now that you have opened a connection, you can create a query and send it to Configuration Server. Starting with release 8.1.4, there are two types of queries supported:

- Filter-based Queries (using `RequestReadObjects`)
- XPath-based Queries (using `RequestReadObjects2`)

If the request is successful, you will receive an `EventObjectsRead` message with the matching data.

> **Tip**
>
> When you send a RequestReadObjects message, Configuration Server may send more than one `EventObjectsRead` messages in response, depending on whether there is too much data to be handled by a single `EventObjectsRead`. Once you have received all of the `EventObjectsRead` messages, Configuration Server will also send an `EventObjectsSent`, which confirms that it has completed your request. For more information, refer to the article on event handling.

Examples of both query types are shown below, showing how you could retrieve information about a particular agent.

Filter-based Queries

For this type of query, you will need to supply the agent's user name using a *filter key*. The filter key tells Configuration Server to narrow your query to a specific agent, rather than retrieving information about all of the persons in your contact center:

```
KeyValueCollection filterKey = new KeyValueCollection();
filterKey.addObject("user_name", userName);
```

You can find the names of the filter keys for Person objects by looking in the *Filter Keys* section of the CfgPerson entry.

> **Tip**

> A similar reference page is available for each Configuration Layer object.

Now you are ready to create the request. For filter-based queries, this is done using `RequestReadObjects`.

As you may know, Configuration Server considers agents to be objects of type CfgPerson. So you will need to create a request for information about a `Person` who has the user name you specified in the filter key:

```
CfgObjectType objectType = CfgObjectType.CFGPerson;
int intPerson = objectType.asInteger();
RequestReadObjects requestFilterQuery =
        RequestReadObjects.create(
                intPerson,
                filterKey);
```

> **Important**
> While the Configuration Layer supports the full character set in defining object names, using certain characters can cause problems in the behavior of some Genesys applications. Avoid using spaces, dashes, periods, or special characters in object names. Consider using underscores where you might normally use spaces or dashes.

After you have created your request, you can send it to Configuration Server, as shown here:

```
confServerProtocol.send(requestFilterQuery);
```

XPath-based Queries

Submitting XPath-based queries is similar to filter-based queries, but does not require any filters or additional objects - instead an XPath search expression is passed to `RequestReadObjects2` as a string.

As in the example above, Configuration Server considers agents to be objects of type CfgPerson. So you will need to create a request for information about a `Person` who has the user name you are looking for:

```
CfgObjectType objectType = CfgObjectType.CFGPerson;
int intPerson = objectType.asInteger();
RequestReadObjects2 requestXpathQuery =
        RequestReadObjects2.create(
                intPerson,
                "CfgPerson[@firstName='John']");
```

> **Important**
> While the Configuration Layer supports the full character set in defining object names,

> using certain characters can cause problems in the behavior of some Genesys applications. Avoid using spaces, dashes, periods, or special characters in object names. Consider using underscores where you might normally use spaces or dashes.

After you have created your request, you can send it to Configuration Server, as shown here:

```
confServerProtocol.send(requestXpathQuery);
```

Interpreting the Response

The 8.5.0 release of Platform SDK introduces new structures for handling configuration object data, instead of the heavyweight DOM trees used in previous Platform SDK releases.

Information that you request is returned by invoking the `getObjects` method of the `EventObjectsRead` message. This method returns a `ConfObject` collection that may contain one or more configuration objects depending on the request query filter.

The `ConfObject` structure is represented as a set of object properties linked with the actual object type metadata description (that is, the schema definition for the configuration server objects).

Using the `toString()` method to dump a sample application object might result in the following:

```
ConfObject(CfgApplication) {
        "DBID" = 631
        "name" = "SampleServerApp-SV-B"
        "type" = 107
        "version" = "8"
        "isServer" = 2
        "serverInfo" = ConfStructure(CfgServerInfo) {
                "hostDBID" = 123
                "port" = "7007"
                "backupServerDBID" = 0
                "timeout" = 10
                "attempts" = 1
        }
        "state" = 1
        "appPrototypeDBID" = 177
        "workDirectory" = "C:\GCTI\SampleServerApp-2\"
        "commandLine" = "SampleServerApp.cmd"
        "autoRestart" = 1
        "startupTimeout" = 90
        "shutdownTimeout" = 90
        "redundancyType" = 2
        "isPrimary" = 1
        "startupType" = 1
        "portInfos" = ConfStructureCollection[1 item(s)] = {
                [0]: ConfStructure(CfgPortInfo) {
                        "id" = "default"
                        "port" = "7007"
                        "longField1" = 0
                        "longField2" = 0
                        "longField3" = 0
                        "longField4" = 0
```

```
                }
        }
        "componentType" = 0
}
```

Actual property values can be get or set using this property schema name, as shown below:

```
Integer objDbid = (Integer) confObject.getPropertyValue("DBID");
String objName = (String) confObject.getPropertyValue("name");

confObject.setPropertyValue("name", objNewName);
```

Configuration protocol metadata descriptions for a configuration object or its properties may be received using the following code:

```
CfgDescriptionObject classInfo = confObject.getClassInfo();
CfgDescriptionAttribute attrInfo = confObject.getPropertyInfo("workDirectory");
```

This API is designed as low-level protocol structures with the ability to support different protocol/schema versions, including forward compatibility features. That is, you can connect to some "future" version of Configuration Server, load its schema information with new objects types and/or with new objects properties, and handle its data in the same way.

Updating an Object

The Configuration Server protocol to update a configuration object uses a special kind of data structure - a "delta object". Delta objects contain object identification (DBID) and new values for changed properties.

Delta objects may be created and filled directly, or with the help of a delta utility (see the `ConfDeltaUtility` class, included in the configuration protocol library).

```
ConfObjectDelta delta0 = new ConfObjectDelta(metadata, CfgObjectType.CFGApplication);
ConfObject inDelta = (ConfObject) delta0.getOrCreatePropertyValue("deltaApplication");
inDelta.setPropertyValue("DBID", objCreated.getObjectDbid());
inDelta.setPropertyValue("name", "ConfObject-new-name");

ConfIntegerCollection delTenants = (ConfIntegerCollection)
        delta0.getOrCreatePropertyValue("deletedTenantDBIDs");
delTenants.add(105);

RequestUpdateObject reqUpdate = RequestUpdateObject.create();
reqUpdate.setObjectDelta(delta0);

Message resp = protocol.request(reqUpdate);
if (resp == null) {
    // timeout
} else if (resp instanceof EventObjectUpdated) {
    // the object has been updated
} else if (resp instanceof EventError) {
    // fail((EventError) resp);
} else {
    // unexpected server response
}
```

Creating a New Object

The new Platform SDK Configuration Protocol data structures allow users to create objects on Configuration Server without using the COM application, as shown below:

```
ConfObject obj0 = new ConfObject(metadata, CfgObjectType.CFGApplication);
obj0.setPropertyValue("name", "ConfObject-App-create-test");
obj0.setPropertyValue("type", CfgAppType.CFGGenericServer.asInteger());
obj0.setPropertyValue("version", "8.5.000.00");

obj0.setPropertyValue("workDirectory", ".");
obj0.setPropertyValue("commandLine", ".");

ConfIntegerCollection tenants = (ConfIntegerCollection)
        obj0.getOrCreatePropertyValue("tenantDBIDs");
tenants.add(101);
tenants.add(105);

ConfStructureCollection connInfos = (ConfStructureCollection)
        obj0.getOrCreatePropertyValue("appServerDBIDs");
ConfStructure connInfo = connInfos.createStructure();
connInfo.setPropertyValue("id", "conn1");
connInfo.setPropertyValue("appServerDBID", getSomeAppDbid(CfgAppType.CFGStatServer));
connInfo.setPropertyValue("connProtocol", "addp");
connInfo.setPropertyValue("timoutLocal",  5);
connInfo.setPropertyValue("timoutRemote", 7);
connInfo.setPropertyValue("mode", 3);
connInfo.setPropertyValue("transportParams", "strings-attr-encoding=utf-8");
connInfo.setPropertyValue("longField1", 0);
connInfos.add(connInfo);

RequestCreateObject reqCreate = RequestCreateObject.create();
reqCreate.setObject(obj0);

Message resp = protocol.request(reqCreate);
if (resp == null) {
    // timeout
} else if (resp instanceof EventObjectCreated) {
    ConfObject objCreated = ((EventObjectCreated) resp).getObject();
    // here we have created object from the server side with assigned DBID
} else if (resp instanceof EventError) {
    // fail((EventError) resp);
} else {
    // error: unexpected server response
}
```

Closing the Connection

Finally, when you are finished communicating with the Configuration Server, you should close the connection, in order to minimize resource utilization:

```
confServerProtocol.close();
```

Working with Delta Objects

When using the Configuration Platform SDK to change attribute values of a configuration object, it is important to understand how "delta structures" work.

A delta structure contains values for each attribute in the configuration object. When a change is requested, a delta object is created that contains values for each attribute. Delta values are initialized to either zero (for integer values) or a null string - defaults that indicate no change should be made for that attribute. To change attributes of a configuration object, you first set the delta value for that attribute and then send the request to Configuration Server to be processed. Only attribute values that are changing should be specified in the delta structure for that object.

Any attributes with a delta value set to zero are left unchanged, so there are two special cases to remember when updating integer values in a configuration object:

- leaving the integer as 0 (zero) means that attribute does not change;
- setting a delta value to the current value of the configuration object attribute will change that attribute value to zero.

For example, if an Agent skill level is currently set to 5, then the following table illustrates the effect of various delta structure values:

Initial Attribute Value	Delta Structure Value	Updated Attribute Value	Comment
5	3	3	Setting the delta structure value to a non-zero integer will change the attribute to that value.
5	0	5	Leaving the delta structure value as zero will leave the attribute unchanged.
5	5	0	Setting the delta structure value to the current attribute value will change the attribute to zero.

Requests sent by SOAP clients and formed in an XML format do not use delta structures, because these types of request do not require all attributes to be present. The COM application block (which is shipped with the Platform SDKs) provides a "non-delta" interface and uses delta objects internally to help users update objects, as shown in the following code snippet:

```
//retrieve an agent that has a single skill, with skill level set to 5
CfgPersonQuery query = new CfgPersonQuery();
query.setUserName("userName");
CfgPerson person = confService.retrieveObject(CfgPerson.class, query);

//Setting the skill level to 5 again will NOT result in a change in skill level  (ie: it will
remain 5).
((List<CfgSkillLevel>)person.getAgentInfo().getSkillLevels()).get(0).setLevel(5);
person.save();
```

```
//Setting the skill level to 0 will actually change the current skill level value.
((List<CfgSkillLevel>)person.getAgentInfo().getSkillLevels()).get(0).setLevel(0);
person.save();
```

To aid you in working with delta objects, Configuration SDK provides the `ConfDeltaUtility` helper class, which contains two public methods:

- `public ConfObjectDelta createDelta(ConfObject actualObj, ConfObject changedObj);`
- `public void applyDelta(ConfObject theObject, ConfObjectDelta delta);`

An instance of this helper class can be created using the actual configuration metadata from an open Configuration Server protocol connection:

```
ConfDeltaUtility deltaUtil = new
ConfDeltaUtility(csProtocol.getServerContext().getMetadata());
```

`ConfObject` objects are cloneable, so it is possible to use the delta utility in the following manner:

```
ConfObject someObject = ...; // get some object from config server
ConfObject objClone = (ConfObject) someObject.clone(); // create a copy

objClone.setPropertyValue("name", "ConfObject-new-name"); // change some property(-ies)
objClone.set...(...);

ConfObjectDelta delta = deltaUtil.createDelta(someObject, objClone);
// use 'delta' to update the object on config server with RequestUpdateObject
```

.NET

Setting Up a ConfServerProtocol Object

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

```
ConfServerProtocol confServerProtocol =
        new ConfServerProtocol(
                new Endpoint(
                        confServerUri));
```

Configuration Server needs some additional information in order to create a successful connection. This information includes the type of client you wish to create, your client's name, and your user name and password:

```
confServerProtocol.ClientApplicationType = (int) CfgAppType.CFGSCE;
confServerProtocol.ClientName = clientName;
confServerProtocol.UserName = userName;
confServerProtocol.UserPassword = password;
```

After instantiating the `ConfServerProtocol` object, you need to open the connection to the Configuration Server:

```
confServerProtocol.Open();
```

Creating a Query

Now that you have opened a connection, you can create a query and send it to Configuration Server. Starting with release 8.1.4, there are two types of queries supported:

- Filter-based Queries (using `RequestReadObjects`)
- XPath-based Queries (using `RequestReadObjects2`)

If the request is successful, you will receive an `EventObjectsRead` message with the matching data.

> **Tip**
> When you send a RequestReadObjects message, Configuration Server may send more than one `EventObjectsRead` messages in response, depending on whether there is too much data to be handled by a single `EventObjectsRead`. Once you have received all of the `EventObjectsRead` messages, Configuration Server will also send an `EventObjectsSent`, which confirms that it has completed your request. For more information, refer to the article on event handling.

Examples of both query types are shown below, showing how you could retrieve information about a particular agent.

Filter-based Queries

For this type of query, you will need to supply the agent's user name using a *filter key*. The filter key tells Configuration Server to narrow your query to a specific agent, rather than retrieving information about all of the persons in your contact center:

```
KeyValueCollection filterKey = new KeyValueCollection();
filterKey.Add("user_name", userName);
```

You can find the names of the filter keys for Person objects by looking in the *Filter Keys* section of the CfgPerson entry.

> **Tip**
> A similar reference page is available for each Configuration Layer object.

Now you are ready to create the request. For filter-based queries, this is done using

RequestReadObjects.

As you may know, Configuration Server considers agents to be objects of type CfgPerson. So you will need to create a request for information about a Person who has the user name you specified in the filter key:

```
RequestReadObjects requestFilterQuery =
        RequestReadObjects.Create(
                (int) CfgObjectType.CFGPerson,
                filterKey);
```

> **Important**
>
> While the Configuration Layer supports the full character set in defining object names, using certain characters can cause problems in the behavior of some Genesys applications. Avoid using spaces, dashes, periods, or special characters in object names. Consider using underscores where you might normally use spaces or dashes.

After you have created your request, you can send it to Configuration Server, as shown here:

```
confServerProtocol.Send(requestFilterQuery);
```

XPath-based Queries

Submitting XPath-based queries is similar to filter-based queries, but does not require any filters or additional objects - instead an XPath search expression is passed to RequestReadObjects2 as a string.

As in the example above, Configuration Server considers agents to be objects of type CfgPerson. So you will need to create a request for information about a Person who has the user name you are looking for:

```
RequestReadObjects2 requestXpathQuery =
        RequestReadObjects2.Create(
                (int) CfgObjectType.CFGPerson,
                "CfgPerson[@firstName='John']");
```

> **Important**
>
> While the Configuration Layer supports the full character set in defining object names, using certain characters can cause problems in the behavior of some Genesys applications. Avoid using spaces, dashes, periods, or special characters in object names. Consider using underscores where you might normally use spaces or dashes.

After you have created your request, you can send it to Configuration Server, as shown here:

```
confServerProtocol.Send(requestXpathQuery);
```

Interpreting the Response

The information you asked for is returned in the `ConfObject` property of the `EventObjectsRead` message.

Here is a sample of how you might print the XML document:

```
EventObjectsRead objectsRead = theMessage;

StringBuilder xmlAsText = new StringBuilder();
XmlWriterSettings xmlSettings = new XmlWriterSettings();
xmlSettings.Indent = true;

using (XmlWriter xmlWriter =
        XmlWriter.Create(xmlAsText, xmlSettings))
{
        XDocument resultDocument = objectsRead.ConfObject;
        resultDocument.WriteTo(xmlWriter);
}

Console.WriteLine("This is the response:\n"
                + xmlAsText.ToString() + "\n\n");
```

And this is what the XML document might look like:

```
 <ConfData>
   <CfgPerson>
     <DBID value="105"/>
     <tenantDBID value="101"/>
     <lastName value="agent1"/>
     <firstName value="Agent"/>
     <employeeID value="agent1"/>
     <userName value="agent1"/>
     <password value="204904E461002B28511D5880E1C36A0F"/>
     <isAgent value="2"/>
     <CfgAgentInfo>
       <placeDBID value="102"/>
       <skillLevels>
         <CfgSkillLevel>
           <skillDBID value="101"/>
           <level value="9"/>
         </CfgSkillLevel>
       </skillLevels>
       <agentLogins>
         <CfgAgentLoginInfo>
           <agentLoginDBID value="103"/>
           <wrapupTime value="0"/>
         </CfgAgentLoginInfo>
       </agentLogins>
       <capacityRuleDBID value="127"/>
     </CfgAgentInfo>
     <isAdmin value="1"/>
     <state value="1"/>
     <userProperties>
       <list_pair key="desktop-redial">
         <str_pair key="phone-number0" value="5551212"/>
         <str_pair key="phone-number1" value=""/>
         <str_pair key="phone-number2" value=""/>
         <str_pair key="phone-number3" value=""/>
         <str_pair key="phone-number4" value=""/>
```

```
        <str_pair key="phone-number5" value=""/>
        <str_pair key="phone-number6" value=""/>
        <str_pair key="phone-number7" value=""/>
        <str_pair key="phone-number8" value=""/>
        <str_pair key="phone-number9" value=""/>
      </list_pair>
      <list_pair key="multimedia">
        <str_pair key="last-media-logged"
            value="voice,email"/>
      </list_pair>
    </userProperties>
    <emailAddress value="agent1@techpubs3"/>
  </CfgPerson>
</ConfData>
```

This XML document contains information about a Person. To interpret the information contained in the document, look at the Parameters section of the CfgPerson entry in the list of Configuration Objects.

If you compare the elements in this XML document to the CfgPerson entry, you can see that some of them contain information that is explained in detail in another entry. For example, the `CfgAgentInfo` element contains information that is described in the CfgAgentInfo entry. Similarly, the `CfgAgentLoginInfo` element contains information described in the CfgAgentLoginInfo entry.

Updating an Object

You can update a Configuration Layer object by passing in an XML document (of type `XDocument`) containing the appropriate information about that object:

```
RequestUpdateObject requestUpdateObject =
        RequestUpdateObject.Create(
                (int) CfgObjectType.CFGPerson,
                xDocument);
```

Creating a New Object

You can also create a new Configuration Layer object by sending an XML Document (of type `XDocument`) to Configuration Server, as shown here:

```
RequestCreateObject requestCreateObject =
        RequestCreateObject.Create(
                (int) CfgObjectType.CFGPerson,
                xDocument);
```

Closing the Connection

Finally, when you are finished communicating with the Configuration Server, you should close the connection, in order to minimize resource utilization:

```
confServerProtocol.Close();
```

Working with Delta Objects

When using the Configuration Platform SDK to change attribute values of a configuration object, it is important to understand how "delta structures" work.

A delta structure contains values for each attribute in the configuration object. When a change is requested, a delta object is created that contains values for each attribute. Delta values are initialized to either zero (for integer values) or a null string - defaults that indicate no change should be made for that attribute. To change attributes of a configuration object, you first set the delta value for that attribute and then send the request to Configuration Server to be processed. Only attribute values that are changing should be specified in the delta structure for that object.

Any attributes with a delta value set to zero are left unchanged, so there are two special cases to remember when updating integer values in a configuration object:

- leaving the integer as 0 (zero) means that attribute does not change;
- setting a delta value to the current value of the configuration object attribute will change that attribute value to zero.

For example, if an Agent skill level is currently set to 5, then the following table illustrates the effect of various delta structure values:

Initial Attribute Value	Delta Structure Value	Updated Attribute Value	Comment
5	3	3	Setting the delta structure value to a non-zero integer will change the attribute to that value.
5	0	5	Leaving the delta structure value as zero will leave the attribute unchanged.
5	5	0	Setting the delta structure value to the current attribute value will change the attribute to zero.

Note that requests sent by SOAP clients and formed in an XML format do not use delta structures, because these types of request do not require all attributes to be present. The COM application block (which is shipped with the Platform SDKs) provides a "non-delta" interface and uses delta objects internally to help users update objects, as shown in the following code snippet:

```
//retrieve a particular agent whose last name is "Jones"
CfgPersonQuery query = new CfgPersonQuery();
query.UserName = "userName";
query.LastName = "Jones";
CfgPerson person = myConfService.RetrieveObject<CfgPerson>(query);

//Setting the last name to the same value will NOT result in a change
person.LastName = "Jones";
person.Save();
```

```
//Setting the last name to a different value will change the actual value
person.LastName = "Smith";
person.Save();
```