Platform SDK Developer's Guide

JSON Support

9/19/2025

JSON Support

Java

Starting with release 8.5.201.04, Platform SDK for Java has been extended with functionality for serialization and deserialization of protocol messages to JSON string representation. (For older 8.5.201.x releases, refer to the Legacy Content section at the bottom of this article.)

Design

The serializer supports two different types of JSON representations for Platform SDK protocol messages: with `MessageName` attribute, and without it.

Adding the `MessageName` attribute helps users to deserialize a protocol message when its type is not known from context. Deserialization without the inner `MessageName` attribute can be used to support existing solutions, or to avoid duplicating some request context data in order to optimize network traffic, CPU, and memory usage.

> **Important**
> Each serializer instance should work with one protocol type. If you need to process different types (such as UCS and StatServer) then you must create one serializer per protocol.

Examples

Using the Platform SDK JSON Message Serializer

> **Tip**
> This serializer is designed to work with Platform SDK messages only. Serialization of other custom classes is not supported.

Example 1: Message Deserialization (with messageName in JSON)

```
PsdkJsonSerializer ser = PsdkJsonSerializer.createContactServerSerializer();
```

```
String json = "{ \"messageName\":\"RequestRefresh\", "
       + "\"query\":\"test-Query-4\", \"file\":\"test-File-3\",\"indexName\":\"Contact\",
\"persistents\":\"test-Persistents-2\"}";

Message message = ser.deserialize(json);
```

Example 2: Message Deserialization (Without messageName in JSON)

```
PsdkJsonSerializer ser = PsdkJsonSerializer.createContactServerSerializer();

String json = "{ \"query\":\"test-Query-4\", \"file\":\"test-
File-3\",\"indexName\":\"Contact\", \"persistents\":\"test-Persistents-2\"}";

Message message = ser.deserialize(json, "RequestRefresh");
// RequestRefresh message = ser.deserialize(json, RequestRefresh.class);
```

Example 3: Message Serialization (With messageName in JSON)

```
PsdkJsonSerializer ser = PsdkJsonSerializer.createContactServerSerializer();

RequestRefresh request = RequestRefresh.create();
request.setQuery("test-Query-4");
request.setFile("test-File-3");
request.setIndexName(IndexNameType.Contact);
request.setPersistents("test-Persistents-2");

String json = ser.serialize(request);
```

Using the Jackson Framework

Example 1: Message Deserialization (with messageName in JSON)

```
ObjectMapper m = new ObjectMapper();
m.registerModule(new ContactServerModule(true));

String json = "{ \"messageName\":\"RequestRefresh\", "
       + "\"query\":\"test-Query-4\", \"file\":\"test-File-3\",\"indexName\":\"Contact\",
\"persistents\":\"test-Persistents-2\"}";

Message message = (Message)m.readValue(json, ContactServerMessage.class);
```

Example 2: Message Deserialization (Without messageName in JSON)

```
ObjectMapper m = new ObjectMapper();
PSDKCommonModule mod = new ContactServerModule();
m.registerModule(mod);

String json = "{ \"query\":\"test-Query-4\", \"file\":\"test-
File-3\",\"indexName\":\"Contact\", \"persistents\":\"test-Persistents-2\"}";

Message message = m.readValue(json, mod.getMessageClass("RequestRefresh"));
//RequestRefresh message = m.readValue(json, RequestRefresh.class);
```

Example 3: Message Serialization (Without messageName in JSON)

```
ObjectMapper m = new ObjectMapper();
m.registerModule(new ContactServerModule());

RequestRefresh request = RequestRefresh.create();
```

```
request.setQuery("test-Query-4");
request.setFile("test-File-3");
request.setIndexName(IndexNameType.Contact);
request.setPersistents("test-Persistents-2");

String json = m.writeValueAsString(request);

// We expect to get JSON like the following:
//  "{ "\"query\":\"test-Query-4\", \"file\":\"test-File-3\",\"indexName\":\"Contact\",
\"persistents\":\"test-Persistents-2\"}";
```

Example 4: Message Serialization (With messageName in JSON)

```
ObjectMapper m = new ObjectMapper();
m.registerModule(new ContactServerModule(true));

RequestRefresh request = RequestRefresh.create();
request.setQuery("test-Query-4");
request.setFile("test-File-3");
request.setIndexName(IndexNameType.Contact);
request.setPersistents("test-Persistents-2");

String json = m.writeValueAsString(request);

// We expect to get JSON like the following:
//  "{ \"messageName\":\"RequestRefresh\", "
//        + "\"query\":\"test-Query-4\", \"file\":\"test-File-3\",\"indexName\":\"Contact\",
\"persistents\":\"test-Persistents-2\"}";
```

Legacy Content

Prior to release 8.5.201.04, Platform SDK for Java did not offer native support for JSON - only XML serialization was supported. This section describes how provide JSON support within your Platform SDK for Java applications for legacy applications that use JSON format for data.

[+] Display Legacy Content

Overview

In most Genesys projects, the Jackson framework is used for serialization/deserialization of plain old Java objects (POJO) in/from JSON. Not all Platform SDK messages are POJO that can be transformed in/from JSON automatically, however.

So to provide full JSON support Platform SDK for Java, a Jackson module is included with your distribution. This module is a separate JAR library; no dependency to the Jackson library has been added directly within Platform SDK.

Using the Module

This section contains code snippets that illustrate how to use the new module.

Example 1: Using the ConfServerModule

```
// create Jackson's JSONizator
```

```
ObjectMapper mapper = new ObjectMapper();
// register our new Platform SDK module
mapper.registerModule( new ConfServerModule() );
// create Platform SDK message
RequestCreateObject request= RequestCreateObject.create();
// serialize message
String json = mapper.writeValueAsString(msg);
// deserialize message
RequestCreateObject msg = objectMapper.readValue(json, RequestCreateObject.class);
```

Example 2: Using a specified set of metadata from Configuration Server

This approach is useful when you are connecting to a more recent version of Configuration Server then Platform SDK supports.

Configuration Server messages that are deserialized using the metadata included with your release of Platform SDK can be sent to older releases of Configuration Server, but in this scenario the unknown (new) attributes for Configuration Server will be ignored while sending the message.

```
ConfServerProtocol c= new ...;
c.open();
CfgMetadata metadata =
((ConfServerContext)c.connectionContext().serverContext()).getMetadata();
// create Jackson's JSONizator
ObjectMapper mapper = new ObjectMapper();
// register our new Platform SDK module
mapper.registerModule( new ConfServerModule(metadata) );
// create Platform SDK message
RequestCreateObject request= RequestCreateObject.create();
// serialize message
String json = mapper.writeValueAsString(msg);
// deserialize message
RequestCreateObject msg = objectMapper.readValue(json, RequestCreateObject.class);
```

Example 3: Configuring Jackson ObjectMapper to support multiple Platform SDK protocols

```
// create Jackson's JSONizator
ObjectMapper mapper = new ObjectMapper();
// register our new Platform SDK modules
mapper.registerModules( new ConfServerModule(), new ContactServerModule() );
mapper.registerModule( new StatServerModule() );
```

Notable Jackson Features

Jackson has many configurable features. This section describes a few features that are particularly noteworthy for Platform SDK developers.

Property Naming

Use this feature carefully, because if you change property naming for serialization then anyone who wants to deserialize content must use the same configuration or else the deserialization will fail. An example of setting a custom property naming strategy is included below:

```
ObjectMapper mapper = new ObjectMapper();
mapper.setPropertyNamingStrategy(PropertyNamingStrategy.LOWER_CASE);
```

Handling Unknown Properties

By default, Jackson fails if a unknown property occurs during deserialization. It is possible to change this behavior so that unknown properties can be ignored, as shown below:

```
ObjectMapper mapper = new ObjectMapper();
mapper.disable(DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES);
mapper.enable(DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES);
```

Relation to Platform SDK ObjectMapper

All Jackson modules included with Platform SDK make changes to the `ObjectMapper` configuration while registering, as shown here:

```
mapper.setSerializationInclusion(Include.NON_NULL);
mapper.configure(SerializationFeature.FAIL_ON_EMPTY_BEANS, false);
mapper.setDateFormat(new ISO8601DateFormatWithMilliseconds());
```

If you want to customer how the `ObjectMapper` will be configured, be sure to apply your changes *after* the Jackson module has registered.

JSON Format Examples

Several examples of the resulting JSON format are provided here for illustrative purposes.

JSON Representation of KVList

```
[ {
  "key" : "int1",
  "type" : "int",
  "value" : 1
}, {
  "key" : "string1",
  "type" : "str",
  "value" : "string-value1"
}, {
  "key" : "utf-string1",
  "type" : "utf",
  "value" : "utf-string-value1"
}, {
  "key" : "binary1",
  "type" : "byte[]",
  "value" : "AQIECP8="
}, {
  "key" : "kv1",
  "type" : "kvlist",
  "value" : [ {
                "key" :"int2",
                "type" : "int",
                "value" : 2
              }, {
                "key" : "string2",
                "type" : "str",
                "value" : "string-value2"
              }, {
                "key" : "utf-string2",
                "type" : "utf",
                "value" : "utf-string-value2"
```

```
            }, {
              "key" : "binary2",
              "type" : "byte[]",
              "value" : "AwQFBr8="
             } ]
```

Notes:

- KeyValuePair property "type" can be skipped for the following types: `int`, `str`, `kvlist` (KeyValueCollection). Platform SDK will deserialize similar data by determining the type heuristically.

JSON Representation of RequestInsertInteraction

```
{
  "interactionAttributes" : {
    "id" : "some-id",
    "entityTypeId" : "Chat",
    "lang" : "en"
  },
  "interactionContent" : {
    "mimeType" : "text/plain",
    "text" : "some text"
  },
  "entityAttributes" : {
    "$type" : ".ChatEntityAttributes",
    "establishedDate" : "2015-07-24",
    "nullAttributes" : [ "ReleasedDate" ]
  }
}
```

Notes:

- $type supports the following values:
  - ".ChatEntityAttributes"
  - ".EmailInEntityAttributes"
  - ".EmailOutEntityAttributes"
  - ".CoBrowseEntityAttributes"
  - ".PhoneCallEntityAttributes"

JSON Representation of RequestContactListGet

```
{
  "contactCount" : true,
  "tenantId" : 15,
  "attributeList" : [ "attr-1", "attr-5", "attr-a" ],
  "sortCriteria" : [ {
    "$type" : ".SimpleSearchCriteria",
    "attrName" : "attr-1",
    "sortIndex" : 0,
    "sortOperator" : "Ascending"
  }, {
    "$type" : ".SimpleSearchCriteria",
    "attrName" : "attr-a",
    "sortIndex" : 1,
    "sortOperator" : "Descending"
```

```
  } ],
  "searchCriteria" : [ {
    "$type" : ".ComplexSearchCriteria",
    "prefix" : "And",
    "criterias" : [ {
      "$type" : ".SimpleSearchCriteria",
      "attrName" : "a",
      "operator" : "Greater",
      "attrValue" : "5"
    } ]
  }, {
    "$type" : ".ComplexSearchCriteria",
    "prefix" : "And",
    "criterias" : [ {
      "$type" : ".SimpleSearchCriteria",
      "attrName" : "b",
      "operator" : "Lesser",
      "attrValue" : "2"
    } ]
  }, {
    "$type" : ".ComplexSearchCriteria",
    "prefix" : "And",
    "criterias" : [ {
      "$type" : ".SimpleSearchCriteria",
      "attrName" : "c",
      "operator" : "Equal",
      "attrValue" : "11"
    } ]
  }, {
    "$type" : ".ComplexSearchCriteria",
    "prefix" : "And",
    "criterias" : [ {
      "prefix" : "Or",
      "criterias" : [ {
        "$type" : ".SimpleSearchCriteria",
        "attrName" : "e",
        "operator" : "Lesser",
        "attrValue" : "1"
      } ]
    }, {
      "$type" : ".ComplexSearchCriteria",
      "prefix" : "Or",
      "criterias" : [ {
        "$type" : ".SimpleSearchCriteria",
        "attrName" : "k",
        "operator" : "GreaterOrEqual",
        "attrValue" : "0"
      } ]
    } ]
  } ]
}
```

Notes:

- $type supports the following values:
  - ".SimpleSearchCriteria"
  - ".ComplexSearchCriteria"

Frequently Asked Questions

Q: The KV lists can contain pointers to outer KVlists, creating circular dependencies. Will such structures be serialized/deserialized?
A: No. Platform SDK does not support sending or receiving such structures.

Q: In what format are binary values serialized? Base64 or something else?
A: Base64

Q: Why is the "type" attribute is optional?
A: The attribute is only optional for the following types: integer, string and key-value collection (it can be skipped when you type JSON requests manually). The Platform SDK Jackson module will always serialize the type attribute to JSON, but can deserialize it without the optional attribute "type" attribute.

Q: Is GEnum serialization/deserialization supported?
A: Yes. As you can see the example above where the GEnum `EntityTypes` is serialized as simple string value "chat".

Q: Does the module perform direct serialization/deserialization or does it use intermediate XML?
A: It performs direct serialization from Platform SDK messages to JSON (and vice versa) using Jackson objectmapper with registered PSDKModule.

.NET

Starting with release 8.5.201.04, Platform SDK for .NET has been extended with functionality for serialization and deserialization of protocol messages to JSON string representation.

Design

The serializer supports two different types of JSON representations for Platform SDK protocol messages: with `MessageName` attribute, and without it.

Adding the `MessageName` attribute helps users to deserialize a protocol message when its type is not known from context. Deserialization without the inner `MessageName` attribute can be used to support existing solutions, or to avoid duplicating some request context data in order to optimize network traffic, CPU, and memory usage.

> **Important**
> Each serializer instance should work with one protocol type. If you need to process different types (such as UCS and StatServer) then you must create one serializer per protocol.

Examples

Example 1: Bi-directional JSON Serialization

```
public void TestRequestGetContacts()
{
  var req = RequestGetContacts.Create();
  req.ReferenceId = 1;
  req.SearchCriteria = new SearchCriteriaCollection();
  req.SortCriteria = new SortCriteriaCollection();
  var json = PsdkJsonSerializer.Serialize(req);
  Console.WriteLine(json);
  var newRq = PsdkJsonSerializer.Deserialize(req.GetType(), json);
  var json2 = PsdkJsonSerializer.Serialize(newRq);
  Console.WriteLine(json2);
  Assert.IsTrue(json.Equals(json2));
}
```

Example 2: Using Platform SDK JSON Serializer for UCS Protocol

```
public void TestRequestGetVersion()
{
  var request = RequestGetVersion.Create();
  request.ReferenceId = 1;
  Console.WriteLine(request);
  var serializer = JsonSerializationFactory.GetSerializer<UniversalContactServerProtocol>();
  var json = serializer.Serialize(request);
  Console.WriteLine(json);
  var deserializedMessage = serializer.Deserialize(json);
  Console.WriteLine(deserializedMessage);
  Assert.IsTrue(request.Equals(deserializedMessage));
}
```

Example 3: Sample JSON Object

Message:

```
'RequestRefresh' ('60')
message attributes:
Query           = test-Query-4
File            = test-File-3
IndexName       = Contact
Persistents     = test-Persistents-2
```

JSON Representation:

```
{"messageName":"RequestRefresh","query":"test-Query-4","file":"test-
File-3","indexName":"Contact","persistents":"test-Persistents-2"}
```

Java and .NET Compatibility Note

The internal implementation of messages from the Configuration Server protocol differs for .NET and Java platforms. Data objects in .NET messages are based on the XDocument class, while Java

message use data classes for mapping of data objects. This means that the result of serialization for the same messages from Java and .NET platforms will be different for the Configuration Server protocol (although bi-directional serialization is supported for both platforms).

All other Platform SDK protocols support cross-platform serialization.