JD Edwards EnterpriseOne Tools

Application Interface Services Client Java API Developers Guide

9.2

JD Edwards EnterpriseOne Tools
Application Interface Services Client Java API Developers Guide

9.2

Part Number: E64173-12

Contents

Preface

Welcome to the JD Edwards EnterpriseOne documentation.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at *http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc* .

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit *http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info* or visit *http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs* if you are hearing impaired.

Related Information

For additional information about JD Edwards EnterpriseOne applications, features, content, and training, visit the JD Edwards EnterpriseOne pages on the JD Edwards Resource Library located at:

http://learnjde.com

Conventions

The following text conventions are used in this document:

Convention	Meaning
Bold	Boldface type indicates graphical user interface elements associated with an action or terms defined in text or the glossary.
Italics	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`Monospace`	Monospace type indicates commands within a paragraph, URLs, code examples, text that appears on a screen, or text that you enter.
> Oracle by Example	Indicates a link to an Oracle by Example (OBE). OBEs provide hands-on, step- by-step instructions, including screen captures that guide you through a process using your own environment. Access to OBEs requires a valid Oracle account.

1 Understanding the AIS Client Java API

Overview

With the Application Interface Services (AIS) Client Java API, you can use any development tool that works with Java APIs to create custom applications that interact with EnterpriseOne. Whether you need a simplified kiosk application for your warehouse, an application that composites features from multiple EnterpriseOne applications into a single purpose-built interface, or an application for the latest wearable device, the AIS Client Java API enables you to choose the development platform that fits your needs.

The AIS Client Java API enables developers to create applications, referred to as AIS clients, that communicate with the JD Edwards EnterpriseOne AIS Server. The AIS Server is a REST services server that when configured with the EnterpriseOne HTML Server, enables access to EnterpriseOne forms and data. The AIS Client Java API provides classes and methods that enable AIS clients to manage (create, read, update, delete) data in EnterpriseOne through REST services.

Note: The EnterpriseOne HTML Server also executes some Java processing; therefore, it is sometimes referred to as the Java Application Server (JAS). The terms HTML Server and JAS Server are synonymous.

Note:

- *"Understanding the JD Edwards EnterpriseOne Application Interface Services (AIS) Server" in the JD Edwards EnterpriseOne Application Interface Services Server Reference Guide* for an overview and illustration of the AIS Server architecture.

Accessing AIS Server Endpoints with the AIS Client Java API

The AIS Server exposes endpoints that:

- Enable access to EnterpriseOne data and applications.
- Produce JSON responses.

Each endpoint provides a particular service, referred to as an AIS service, that AIS clients can use to interact with EnterpriseOne applications. The AIS Client Java API enables easy access to all endpoints; all of the communication is handled for you.

Starting with EnterpriseOne Tools release 9.2.1.2, the AIS Server provides version 2 AIS services that include additional capabilities for AIS clients. All AIS services that were available before version 2 are also available as version 2 AIS services. See *"AIS Services (Endpoints)" in the JD Edwards EnterpriseOne Application Interface Services Server Reference Guide* for a list of the AIS Server endpoints and a description of the AIS service each endpoint provides.

You must use the AIS Client Java API 2.0.0 to access version 2 AIS services. Use the following URL format to access the endpoints for version 2 AIS services: `http://<server>:<port>/jderest/v2/<URI>`

Note: AIS Java Client API version 1.4.2 is still available and compatible with previous AIS Server releases.

All POST calls expect JSON formatted request payloads.

When you use the API, you work with Java objects, not the JSON strings. But it is still important to understand how the data is transmitted. The following chapters in this guide describe in detail how to use the services the endpoints provide and the Java objects required to use them:

- *Configuring the Login Environment*
- *Performing AIS Form Service Calls*

2 Getting Started

Certifications (Formerly Known as Minimum Technical Requirements)

Customers must conform to the supported platforms for the release, which can be found in the Certifications tab on My Oracle Support: *https://support.oracle.com* .

For more information about JD Edwards EnterpriseOne Minimum Technical Requirements, see the following document on My Oracle Support: JD Edwards EnterpriseOne Minimum Technical Requirements Reference (Doc ID 745831.1), which is available here:

https://support.oracle.com/rs?type=doc&id=745831.1

Prerequisites

To develop AIS client applications, you must complete the following prerequisites:

- You must be running a minimum of JD Edwards EnterpriseOne Tools release 9.1.5.
- Deploy an Application Interface Service (AIS) Server configured with an EnterpriseOne HTML Server. See:

  "Create an Application Interface Services (AIS) Server as a New Managed Instance" in the JD Edwards EnterpriseOne Tools Server Manager Guide .

  and

  "Configuring the AIS Server" in the JD Edwards EnterpriseOne Application Interface Services Server Reference Guide for additional configuration steps.

- Download the latest AIS_Client_Java_API_2.x.x from the JD Edwards Update Center on My Oracle Support: *https://updatecenter.oracle.com/*

  To locate the download on the JD Edwards Update Center, use the Type field to search on "EnterpriseOne ADF."

  The zip file contains:

  - AIS_Client.jar, which contains the AIS Client Java API.

    Click the following link to access the *JD Edwards EnterpriseOne Application Interface Services (AIS) Client API Reference*, which provides descriptions of the AIS Client Java API classes and methods:

    http://docs.oracle.com/cd/E53430_01/nav/development.htm
  - Jackson 2.9.3 library, which includes the jackson-databind, jackson-core, and jackson-annotations jar files.
  - AISCGE 12c_v1.6.x.zip, which contains the AIS Client Class Generator extension for JDeveloper.

    The AIS Client Class Generator is compatible only with JDeveloper 12.1.3 and up. After you download it, see *"Using the AIS Client Class Generator" in the JD Edwards EnterpriseOne Application Interface Services Server Reference Guide* for instructions on how to install and use the AIS Client Class Generator.

Note: The AIS client and Jackson jar files must be in the classpath of your AIS client.

3 Configuring the Login Environment

Configuring the Login

For an AIS client to call AIS services, the AIS client must first obtain a login environment by passing the following information to the constructor in the LoginEnvironment object:

- **EnterpriseOne login credentials**. EnterpriseOne credentials include a user ID, password, environment, and role. The AIS Server configuration uses a default EnterpriseOne environment and role unless you specify a different environment and role here.
- **AIS Server URL and the device name**. The device name is a string that represents the device on which the client is running. The device name serves as a unique identifier for your client.
- **A list of required capabilities**. (Optional) If the AIS client uses AIS Server capabilities, then you have the option to pass a list of required capabilities to the LoginEnvironment constructor. The LoginEnvironment constructor verifies that the capabilities are available on the AIS Server. If they are available, access to the AIS client is granted. If they are not available, access is denied.

  This prevents an AIS client from running if the AIS Server capability that it requires to properly function is not available in the version of the AIS Server. See *Understanding AIS Server Capabilities* for a list of AIS Server capabilities available by EnterpriseOne Tools release.

When the client requests a LoginEnvironment, the processing within the API uses the defaultconfg and tokenrequest URI endpoints. For a description of these endpoints, see the *"AIS Services (Endpoints)" section in the JD Edwards EnterpriseOne Application Interface Services Server Reference Guide* .

Example - Examples for Obtaining a Login Environment

```
//login with minimum required information
final String AIS_SERVER = "http://ais.example.com:7777";
final String USER_NAME = "jde";
final String PASSWORD = "jde";
final String DEVICE = "Java";
LoginEnvironment  loginEnv = new LoginEnvironment(AIS_SERVER, USER_NAME, PASSWORD,
 DEVICE);

//login overrides default environment and role
final String ENVIRONMENT = "PROD";
final String ROLE = "PROLE";
LoginEnvironment  loginEnv2 = new LoginEnvironment(AIS_SERVER, USER_NAME, PASSWORD,
 ENVIRONMENT, ROLE, DEVICE);

//login with required capabilities
//A CapabilityException will be thrown if AIS doesn't have those in the list
final String REQ_CAPABILITIES = "grid, processingOption";
LoginEnvironment  loginEnv3 = new LoginEnvironment(AIS_SERVER, USER_NAME, PASSWORD,
 DEVICE, REQ_CAPABILITIES);

//login with token
String PS_TOKEN = "a ps token string";
LoginEnvironment  loginEnv4 = new LoginEnvironment(AIS_SERVER, USER_NAME, null, null,
 null, DEVICE, null, null, PS_TOKEN)
```

All calls to the AIS Server include the LoginEnvironment object. From this point forward in this guide, references to the loginEnv variable assume that this step has been performed and that the variable is available.

Configuring the Logout

When finished making calls to the AIS Server, you must include the following logout call to end the user session:

```
AISClientUtilities.logout(loginEnv);
```

4 Performing AIS Form Service Calls

Understanding AIS Server Capabilities

The AIS Server exposes various capabilities that AIS client applications may or may not depend on. If your application requires a certain capability, you must include it in the list of required capabilities in the LoginEnvironment constructor.

If you included a capability in the list, the Login module verifies that capability is available when the application launches. If the capability is not available, the application returns an error message. If the capability is available, the application continues to the login screen. See *Configuring the Login Environment* for more information.

You can access the AIS Server capabilities using the following URL:

http://<AIS Server>:<Port>/jderest/defaultconfig

You can also find a description of all AIS Server capabilities in the *"AIS Server Capabilities and Services" section in the JD Edwards EnterpriseOne Application Interface Services Server Reference Guide* .

The code in *Example - Capabilities in LoginEnvironment Constructor* shows the grid and editable capabilities listed in the LoginEnvironment constructor.

Example - Capabilities in LoginEnvironment Constructor

```
            final String REQUIRED_CAP_LIST = "grid,query";
            th.loginEnv = new LoginEnvironment(AIS_SERVER, USER_NAME, PASSWORD,
 ENVIRONMENT, ROLE, DEVICE, REQUIRED_CAP_LIST, JAS_SERVER);
```

If the list includes a capability that is not available on the AIS Server, it throws a CapabilityException, as shown in *Example - Capability Exception*.

Example - Capability Exception

```
com.oracle.e1.aisclient.CapabilityException: Required Capabilities [grid, somethingelse]
 Available Capabilities: [grid, editable, log, processingOption, ignoreFDAFindOnEntry,
 selectAllGridRows, applicationStack, thumbnailSize, gridCellClick, query,
 taskAuthorization, urlMediaObjects, jargon, aliasNaming]
```

Understanding the AIS Client Class Generator

The AIS Client Class Generator is an extension to JDeveloper that enables you to create Application Controller foundational classes that are required by AIS client applications.

For more information about the AIS Client Class Generator, see *"Using the AIS Client Class Generator" in the JD Edwards EnterpriseOne Application Interface Services Server Reference Guide* .

Understanding Form Service Requests

This section contains the following topics:

Overview

AIS Server calls that retrieve data from forms in the EnterpriseOne web client are referred to as form service requests. AIS client applications use form service requests to interact with EnterpriseOne web client forms. Form service requests, formatted as REST service calls that use POST, contain form service events or commands that invoke actions on an EnterpriseOne form.

A form service request enables you to perform various operations on a single form. By sending an ordered list of commands, a form service request can replicate the actions taken by an EnterpriseOne web client user, including populating fields, pressing buttons, and other actions.

To send a form service request to the AIS Server, send a POST to the following URL and send JSON in the body:

```
http://<AIS Server>:<Port>/formservice
```

If testing with a REST testing tool, you can send JSON directly.

The following list is an example of the operations required to perform a query in the find/browse form of the Address Book application (P01012_W01012B):

1. Enter a value into the Search Type field and into the QBE field for address number.
2. Click the check boxes to show the extra grid columns and press the **Find** button.

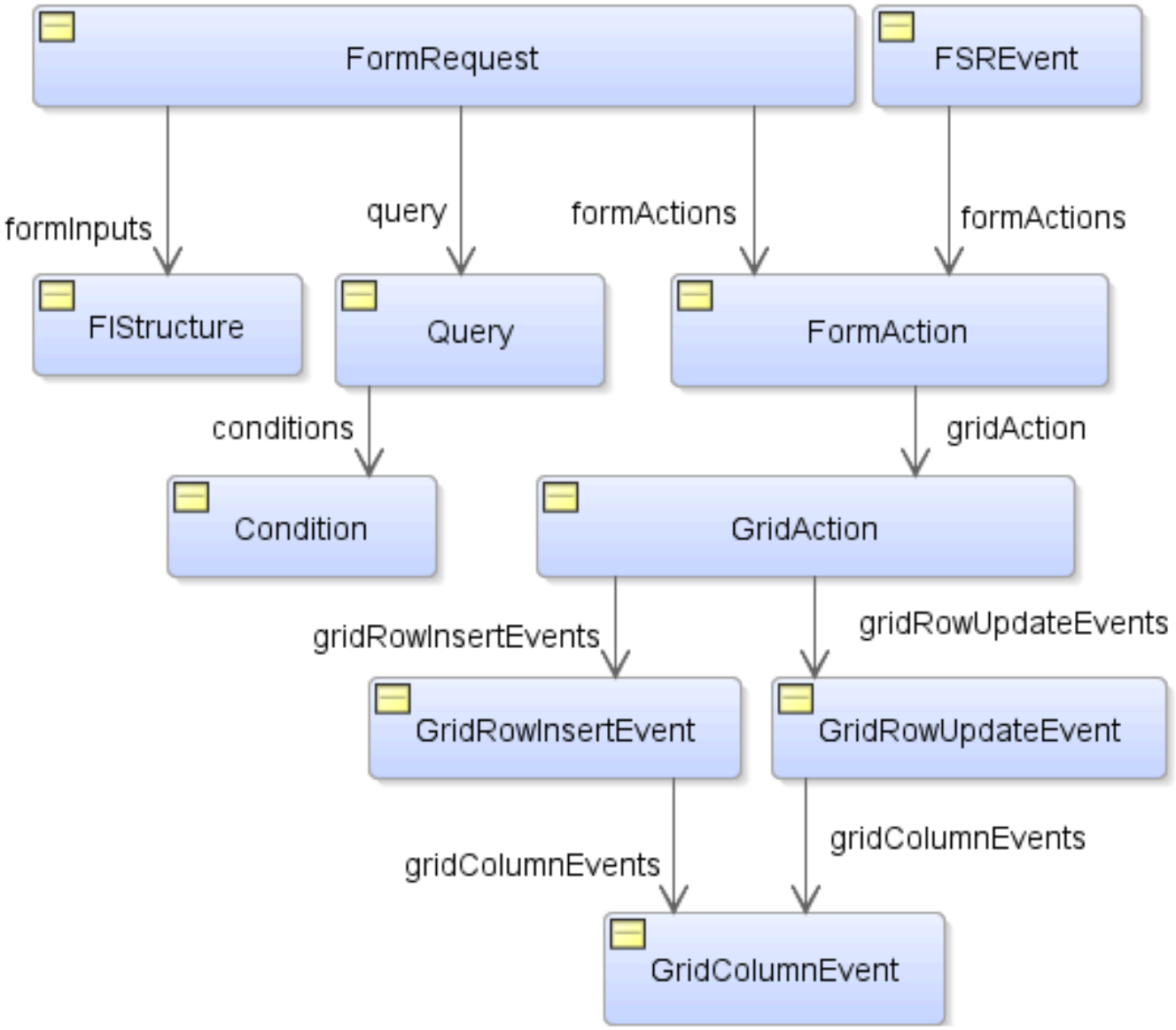
   This populates the grid with the data matching the query.

   The form service returns the form parent object representing the form after it is populated with the data.

Form Service Request Structure

The class diagram in *#unique_25/unique_25_Connect_42_BABHFBIC* represents the basic structure of a form service request. The collections under FormRequest are optional (0 to many); you do not have to have FIStructures,

FormActions, GridActions, and so forth. A form service request (FSR) event is a set of FormActions that you first compile into an FSR event and then add to the FormRequest using the add method.



Control ID Notation for Return Control IDs

In EnterpriseOne, you can use the Property Browser in FDA to identify control IDs for fields on each EnterpriseOne form. You can also find control IDs using the Item Help option in the form in the EnterpriseOne web client. In the EnterpriseOne web client form, click the Help button (question mark in the upper right corner of a form) and then click

the Item Help option to access field-level help. With the field level help activated, you can click in a field or column to access the control ID and business view information, which is displayed under the Advanced Options section.

For fields on the main form, the control ID will be a single value, such as 25.

Grids also have control IDs. For a traditional form, the grid ID is usually 1. For power forms, subforms, and reusable subforms the grid ID is typically a value other than 1.

The columns within a grid also have unique IDs and are often referenced in conjunction with the grid ID. For example, column 28 and 29 in grid 1 would be 1[28,29].

Power forms have more complex IDs. The fields on the main power form are represented with single values. The fields on a subform are complex with an underscore separating them. So field 6 on subform 12 is 12_6. The ID of a re-usable subform is available when viewing the power form that the subform is used on. The IDs of individual fields, a grid, or columns on a re-usable subform is shown in FDA when viewing the subform directly; you cannot get these values when viewing the subform alias.

The returnControlIDs string is bar delimited, without a starting or ending bar.

Example - Requesting fields and grid columns on a traditional form.

```
formRequest.setReturnControlIDs("19|20|60|125|1[45,49,88]");
```

In this example, `19|20|60|125` represent field control IDs.

`1[45,49,88]` represents columns in the grid.

Example - Requesting main form fields, subform fields, main form grid columns, and subform grid columns.

```
formRequest.setReturnControlIDs("33|34|17[24,26,28]|50_45|50_53|50_9[35,39,41]");
```

In this example, `33|34` represent fields on the main form.

`50_45|50_53` represent fields on the subform.

`17[24,26,28]` represent main form grid columns.

`50_9[35,39,41]` represent subform grid columns.

Reading Data

The code in *Example - Form Service Request for Reading Data* is an example of a form service request that reads data from and EnterpriseOne form. In this example, the code results in populating the P01012 form parent object with data that can be displayed or manipulated.

Example - Form Service Request for Reading Data

```
public P01012_W01012B_FormParent P01012()
{
P01012_W01012B_FormParent p01012form = null;

   try{
 //populate the request information
FormRequest formRequest = new FormRequest(loginEnv);
```

```
formRequest.setFormName("P01012_W01012B");
formRequest.setFormServiceAction("R");
formRequest.setMaxPageSize("30"); //only get 30 records
formRequest.setReturnControlIDs("54|1[19,20]");

FSREvent fsrEvent = new FSREvent();


fsrEvent.setFieldValue("54", "E"); //customers
//include >= operator in QBE
fsrEvent.setQBEValue("1[19]", ">=" + "6001");
fsrEvent.checkBoxChecked("62"); //show address
fsrEvent.checkBoxChecked("63"); //show phone
fsrEvent.doControlAction("15"); //find

formRequest.addFSREvent(fsrEvent); //add the events to the request
String response = JDERestServiceProvider.jdeRestServiceCall(loginEnv, formRequest,
 JDERestServiceProvider.POST_METHOD, JDERestServiceProvider.FORM_SERVICE_URI);

 //de-serialize the JSON string into the form parent object
p01012form = loginEnv.getObjectMapper().readValue(response,
 P01012_W01012B_FormParent.class);
  }
  catch(JDERestServiceException e)
  {
      //get more specific error string
      String error = JDERestServiceProvider.handleServiceException(e);
     System.out.println(error);
  }
  catch(Exception e)
  {
      //handle other exceptions
      System.out.println(e.getMessage());
  }


return p01012form;
}
```

Adding Data

The code in *Example - Form Service Request for Adding Data* is an example of a form service request that adds a new phone number in the EnterpriseOne phones application and saves it. After saving the phone number, the form service sends a response with the new number in the grid.

Example - Form Service Request for Adding Data

```
public P0115_W0115A_FormParent addPhone(){
P0115_W0115A_FormParent p0115_W0115A = null;

 //indicate using grid capability
//(alternatively could use required capability)
loginEnv.getUsedCapabilities().add("grid");

if (AISClientCapability.isCapabilityAvailable(loginEnv, "grid"))
```

```
{
  try{

FormRequest formRequest = new FormRequest(loginEnv);
formRequest.setFormName("P0115_W0115A");
formRequest.setFormServiceAction("U");

  //open this form with specific record for AB 7500, contact 0
formRequest.addToFISet("4", "7500");
formRequest.addToFISet("5", "0");

FSREvent fsrEvent = new FSREvent();
//create grid action
GridAction gridAction = new GridAction(loginEnv);
//create grid row insert event
GridRowInsertEvent gri = new GridRowInsertEvent();

//set the column values
gri.setGridColumnValue("27", "HOM");
gri.setGridColumnValue("28", "303");
gri.setGridColumnValue("29", "123-4567");

//add the row to grid ID "1"
gridAction.insertGridRow("1", gri);

//add the grid action to the events
fsrEvent.addGridAction(gridAction);

//press OK button
fsrEvent.doControlAction("4");

//add the FSR event to the request
formRequest.addFSREvent(fsrEvent);


String response = JDERestServiceProvider.jdeRestServiceCall(loginEnv, formRequest,
 JDERestServiceProvider.POST_METHOD, JDERestServiceProvider.FORM_SERVICE_URI);


//de-serialize the JSON string into the form parent object
p0115_W0115A = loginEnv.getObjectMapper().readValue(response,
 P0115_W0115A_FormParent.class);
         }
         catch(CapabilityException e)
         {
//handle capability exception
            System.out.println(e.getMessage());
         }
         catch(JDERestServiceException e)
         {
  //get more specific error string
            String error = JDERestServiceProvider.handleServiceException(e);
            System.out.println(error);
         }
}

 return p0115_W0115A;
}
```

Deleting Data

The code in *Example - Form Service Request for Deleting Data* is an example of a form service request that deletes the phone at index 0 and returns a response with a set of records without the removed phone number record.

Example - Form Service Request for Deleting Data

```
public P0115_W0115A _FormParent deletePhone(){

  P0115_W0115A _FormParent p0115_W0115A = null;
   try{

FormRequest formRequest = new FormRequest(loginEnv);
formRequest.setFormName("P0115_W0115A");
formRequest.setFormServiceAction(formRequest.ACTION_UPDATE);

 //open form with record for AB 7500 contact 0
formRequest.addToFISet("4", "7500");
formRequest.addToFISet("5", "0");

FSREvent fsrEvent = new FSREvent();

//select the row to delete from grid with ID "1", based on row index 0
fsrEvent.selectRow("1", 0);

//press Delete button
fsrEvent.doControlAction("59");

//press OK button
fsrEvent.doControlAction("4");

//add the FSR event to the request
formRequest.addFSREvent(fsrEvent);


String response = JDERestServiceProvider.jdeRestServiceCall(loginEnv, formRequest,
 JDERestServiceProvider.POST_METHOD, JDERestServiceProvider.FORM_SERVICE_URI);


//de-serialize the JSON string into the form parent object
p0115_W0115A = loginEnv.getObjectMapper().readValue(response,
 P0115_W0115A_FormParent.class);
  }
  catch(JDERestServiceException e)
  {
      //get more specific error string
      String error = JDERestServiceProvider.handleServiceException(e);
     System.out.println(error);
  }
  catch(Exception e)
  {
      //handle other exceptions
      System.out.println(e.getMessage());
  }
```

```
return p0115_W0115A;

}
```

Placing Events in the Proper Order

Place the events in the request in the order you want them to execute, for example, populate a filter field value and then press the Find button. Remember that the FDA Form Service Request event occurs before the events you add to this list. Do not set the Find On Entry option when using the event model; the extra "find" is not necessary because it executes before the events you requested.

Considering Hidden Filters and Hidden QBE

By default, values are not written to hidden filter fields or hidden QBE columns. You must use the Form Service Event in FDA to show the fields and columns first. Then you can add values to these fields and subsequently run the query.

Available Actions or Events

The preceding examples in this chapter only show some of the operations you can perform in a form service request. The tables in this section describe other operations you may want to perform.

Form Service Request Events

Action or Event	Description	Parameters
Set Control Value	Sets the value of a control on a form, like filter fields or any other form control.	controlID ("25") value("Bob" or "01/01/2015")
Set QBE Value	Sets the value of a QBE column.	controlID ("1[42]" or "1_2[25]") value ("Jill" or "55")
Set Checkbox Value	Sets the value of a check box.	controlID ("77") value ("on" or "off")
Set Radio Button	Sets the value of a radio button.	controlID ("87") value ("87")
Set Combo Value	Sets the value of a combo box entry.	controlID ("125") value (2) - The index of the entry.

Action or Event	Description	Parameters
Do Action	Presses a button or Hyper Item.	controlID ("156")
Select Row	Selects the specified row in a grid.	controlID ("1.30") - The grid ID, dot, then a row index (zero based).
Select All Rows	Select all rows in the specified grid (if multiple selection is allowed).	controlID ("1") - The grid ID.
Un Select All Rows	Un-selects all rows in the specified grid (if multiple selection is allowed).	controlID ("1") - The grid ID.
Un Select Row	Un-selects the specified row in a grid.	controlID ("1.30") - The grid ID, dot, then a row index (zero based).
Click Grid Cell	Clicks the hyperlink in a grid cell (if the cell is enabled as a link).	controlID ("1.5.22") - The grid ID, dot, row index, dot, grid column ID.
Click Grid Column Aggregation (Available starting with EnterpriseOne Tools 9.2.0.2.)	Clicks the icon for aggregation of a column (if available in the application).	"command": "ClickGridColumnAggregate", controlID ("1.24") – The grid ID, dot, then grid column ID. You must include the capability in the used capability list in order to perform this action.
Next Grid Page (Available starting with EnterpriseOne Tools 9.2.1.)	Clicks the **>** icon on the grid so the next set of records can be returned. This is especially useful in an application stack call in which the application stays open and you can keep retrieving additional records.	"command":"NextGrid", "controlID":"1"

In addition to interacting with fields on the form, you can interact with grids using grid action events. If you use a grid action event, you must include "grid" as a required capability in the LoginEnvironment constructor. See *Understanding AIS Server Capabilities* for more information.

The types of grid action events include:

- Selecting grid rows

  This action enables you to delete records in the grid by sending a row select event, followed by a delete button press event, and then finally an OK button press event. This is the exact sequence that a user would follow to delete a record in an EnterpriseOne application.

- Inserting grid rows

  This action enables you to insert one or more rows into a grid, setting the column value for each row. This includes text entry columns, drop-down columns, or check box columns. You must include an OK button pressed event to commit the inserts.

- Updating grid rows

  This action enables you to update one or more existing grid rows by setting the column values for each row. This includes text entry columns, drop-down columns, or check box columns. You must include an OK button pressed event to commit the updates.

The following table describes the commands that you can use in grid column events to set values for a cell in a grid insert or update event:

Grid Column Events in a Form Service Request

Grid Column Events	Description	Parameters
Set Grid Cell Value	Sets the value of a cell in a grid.	"value": "720", "command": "SetGridCellValue", "columnID": "28"
Set Grid Combo Value	Sets the value of a dropdown column in a grid. The value you send is the 'Code' for the UDC associated with that column.	"value": "ABC", "command": "SetGridComboValue", "columnID": "43"

Determining the Maximum Records Returned in a Form Service Request

In a form service request that returns rows in a grid, the AIS Server will return a maximum of 100 rows by default. If you want to return all records, include the following method in the form service request:

formRequest.setMaxPageSizeUnlimited();

Using Turbo Mode (API 1.4.2 and EnterpriseOne Tools 9.2.1)

To increase transaction performance and reduce form service request processing time on the AIS Server, you can add the Turbo Mode parameter to a form service request. There are two levels for Turbo Mode: Low and High.

Setting the Turbo Mode to "Low" reduces processing time by:

- Fetching associated descriptions in the grid (for grid columns) only when specifically requested in the service request.
- Formatting grid columns only when requested.
- Not using associated descriptions in event rule code.
- Adding columns to return column IDs only when required by internal processing.

Setting the Turbo Mode to "High" provides the best performance and further reduces processing time by:

- Populating business view grid columns only when requested.
- Creating internal grid cell structures only for requested columns.
- Adding columns to return column IDs only when required by internal processing.

> **CAUTION:** Using Turbo Mode can result in issues in the AIS client depending on the processing that is removed. You must make sure that the reduced processing does not impact the data returned to your application. Also, if you are relying on calculated fields, you need to request both the calculated fields and the fields used in the calculation in the return control IDs.

The following code shows the available Turbo Mode options in the API:

```
FormRequest formRequest = new FormRequest(loginEnv);
formRequest.setTurboMode(FormRequest.TURBO_MODE_HIGH);
formRequest.setTurboMode(FormRequest.TURBO_MODE_LOW);
```

The following is an example of Turbo Mode in JSON code:

```
{
"formName": "P01012_W01012";
"turboMode ": "Low";

}
```

Batch Form Service

If you make several sequential calls to forms without any data dependencies between them, consider using the Batch Form Service. Batch form service requests are used to execute multiple EnterpriseOne forms during a single request, which improves your AIS client's performance.

Use the AIS Client Class Generator to generate the classes for all the forms that you need to call in the batch request. Then declare a parent class that contains all of the same forms in the order in which they appear in the batch request (including an index number).

Example - Batch Form Service Request shows a batch form service request that calls the same form three times with different inputs each time, followed by a call to another form.

Example - Batch Form Service Request

```
public class BatchRequestParent {

    private P54HS220_W54HS220A fs_0_P54HS220_W54HS220A;
    private P54HS220_W54HS220A fs_1_P54HS220_W54HS220A;
    private P54HS220_W54HS220A fs_2_P54HS220_W54HS220A;
    private P54HSPT_S54HSPTA fs_3_P54HSPT_S54HSPTA;

    public BatchRequestParent() {
        super();
    }
```

```
    public void setFs_0_P54HS220_W54HS220A(P54HS220_W54HS220A fs_0_P54HS220_W54HS220A) {
        this.fs_0_P54HS220_W54HS220A = fs_0_P54HS220_W54HS220A;
    }

    public P54HS220_W54HS220A getFs_0_P54HS220_W54HS220A() {
        return fs_0_P54HS220_W54HS220A;
    }

    public void setFs_1_P54HS220_W54HS220A(P54HS220_W54HS220A fs_1_P54HS220_W54HS220A) {
        this.fs_1_P54HS220_W54HS220A = fs_1_P54HS220_W54HS220A;
    }

    public P54HS220_W54HS220A getFs_1_P54HS220_W54HS220A() {
        return fs_1_P54HS220_W54HS220A;
    }

    public void setFs_2_P54HS220_W54HS220A(P54HS220_W54HS220A fs_2_P54HS220_W54HS220A) {
        this.fs_2_P54HS220_W54HS220A = fs_2_P54HS220_W54HS220A;
    }

    public P54HS220_W54HS220A getFs_2_P54HS220_W54HS220A() {
        return fs_2_P54HS220_W54HS220A;
    }

    public void setFs_3_P54HS230_W54HS230A(P54HS230_W54HS230A fs_3_P54HS230_W54HS230A) {
        this.fs_3_P54HS230_W54HS230A = fs_3_P54HS230_W54HS230A;
    }

    public P54HS230_W54HS230A getFs_3_P54HS230_W54HS230A() {
        return fs_3_P54HS230_W54HS230A;
    }

    public void setFs_4_P54HS240_W54HS240A(P54HS240_W54HS240A fs_4_P54HS240_W54HS240A) {
        this.fs_4_P54HS240_W54HS240A = fs_4_P54HS240_W54HS240A;
    }

    public P54HS240_W54HS240A getFs_4_P54HS240_W54HS240A() {
        return fs_4_P54HS240_W54HS240A;
    }

    public void setFs_3_P54HSPT_S54HSPTA(P54HSPT_S54HSPTA fs_3_P54HSPT_S54HSPTA) {
        this.fs_3_P54HSPT_S54HSPTA = fs_3_P54HSPT_S54HSPTA;
    }

    public P54HSPT_S54HSPTA getFs_3_P54HSPT_S54HSPTA() {
        return fs_3_P54HSPT_S54HSPTA;
    }
}
```

Example - Deserialize the Response to the BatchRequestParent

This sample code shows how after calling forms, you can call the service and deserialize the response to the BatchRequestParent.

```
public BatchRequestParent batcRequest(){

  BatchRequestParent batchParent = null;
   try{
```

```
        // Get resource bundle for incident category text
        BatchFormRequest batchFormRequest = new BatchFormRequest(loginEnv);


        //recentIncidents - Index 0
        SingleFormRequest formRequest = new SingleFormRequest();
        //formRequest.setFindOnEntry("TRUE");

formRequest.setReturnControlIDs("1[19,20,21,27,28,41,45,46,47,48,49,50,51,52,54,55,92,174,177,178,181]");
        formRequest.setFormName("P54HS220_W54HS220A");

        //create event holder
        FSREvent recentFSREvent = new FSREvent();
        //add filter actions in order
        // Incident From Date
        recentFSREvent.setFieldValueDate(loginEnv, "150", cal.getTime());
        // Potential Incident
        recentFSREvent.setQBEValue("1[30]", "0");
        // Exclude from Safety Statistics
        recentFSREvent.setQBEValue("1[39]", "0");
        // Press Find Button
        recentFSREvent.doControlAction("15");
        //add event holder to the form request
        formRequest.addFSREvent(recentFSREvent);

        batchFormRequest.getFormRequests().add(formRequest);


        //recentInjuryIllnessIncidents  - Index 1
        formRequest = new SingleFormRequest();
        //formRequest.setFindOnEntry("TRUE");

formRequest.setReturnControlIDs("1[19,20,21,27,28,41,45,46,47,48,49,50,51,52,54,55,92,174,177,178,181]");
        formRequest.setFormName("P54HS220_W54HS220A");

        //create event holder
        FSREvent injuryFSREvent = new FSREvent();
        //add filter actions in order
        // Incident From Date
        injuryFSREvent.setFieldValueDate(loginEnv, "150", cal.getTime());
        // Potential Incident
        injuryFSREvent.setQBEValue("1[30]", "0");
        // Exclude from Safety Statistics
        injuryFSREvent.setQBEValue("1[39]", "0");
        // Injury/Illness checkbox
        injuryFSREvent.setQBEValue("1[33]", "1");
        // Press Find Button
        injuryFSREvent.doControlAction("15");
        //add event holder to the form request
        formRequest.addFSREvent(injuryFSREvent);

        batchFormRequest.getFormRequests().add(formRequest);

        // recentEnvironmentalIncidents - Index 2
        formRequest = new SingleFormRequest();
        //formRequest.setFindOnEntry("TRUE");

formRequest.setReturnControlIDs("1[19,20,21,27,28,41,45,46,47,48,49,50,51,52,54,55,92,174,177,178,181]");
        formRequest.setFormName("P54HS220_W54HS220A");

        //create event holder
        FSREvent environFSREvent = new FSREvent();
        //add filter actions in order
        // Incident From Date
        environFSREvent.setFieldValueDate(loginEnv, "150", cal.getTime());
        // Potential Incident
        environFSREvent.setQBEValue("1[30]", "0");
        // Exclude from Safety Statistics
        environFSREvent.setQBEValue("1[39]", "0");
        // Environmental checkbox
        environFSREvent.setQBEValue("1[34]", "1");
        // Press Find Button
        environFSREvent.doControlAction("15");
        //add event holder to the form request
        formRequest.addFSREvent(environFSREvent);
```

```
        batchFormRequest.getFormRequests().add(formRequest);


        // scoreboard - Index 3
        formRequest = new SingleFormRequest();
        formRequest.setFindOnEntry("TRUE");
        formRequest.setReturnControlIDs("1_20|1_22");
        formRequest.setFormName("P54HSPT_S54HSPTA");
        batchFormRequest.getFormRequests().add(formRequest);


        String response = JDERestServiceProvider.jdeRestServiceCall(loginEnv, batchFormRequest,
 JDERestServiceProvider.POST_METHOD, JDERestServiceProvider.BATCH_FORM_SERVICE_URI);

//de-serialize the JSON string into the batchParent object
batchParent = loginEnv.getObjectMapper().readValue(response, BatchRequestParent.class);


}
  catch(JDERestServiceException e)
  {
      //get more specific error string
      String error = JDERestServiceProvider.handleServiceException(e);
     System.out.println(error);
  }
  catch(Exception e)
  {
      //handle other exceptions
      System.out.println(e.getMessage());
  }

  return batchParent;
}
```

Application Stack Service

The application stack service enables an AIS client to interact with multiple applications running in an ongoing EnterpriseOne web client session. The application stack service enables more complex interactions with applications that have cross-form transaction boundaries, for example where you do not want to save the header until the details are added.

The application stack service supports form interconnects in EnterpriseOne to receive data from the resulting form. For example, you may want to use an existing sequence of tasks in EnterpriseOne that involves interacting with multiple forms to perform a transaction: open an initial form; select a record and navigation to a second form; perform an update that might automatically flow to a third form where you enter more data; and then finally complete the transaction. The application stack service allows for this type of interaction with EnterpriseOne forms.

To use the application stack service, you must first create an ApplicationStack object, which contains these three types of operations:

- **Open**. Open starts a new stack, opening the first form and performing any operations included in the FormRequest.
- **Execute**. Subsequent actions on that application stack must use the Execute operation, where you can pass an ActionRequest with any actions to be performed on the currently open form.
- **Close**. You can pass a Close operation to close the stack and any open forms on it.

Each response to a stack request includes the current form which might be the form originally requested, or it could be a new form if navigation to a new form occurred.

Make sure that you are executing actions on the right form. You should use the getLastAppStackResponse().checkSuccess method before executing actions so you can be sure of the current form.

You must include the form in the request for actions. If the form in the request does not match the current form on the stack, the actions will not execute.

The sample code in *Example - Application Stack* performs operations in a stack of applications in this order:

1. Opens the stack first with the Address Book find/browse form (P01012_W0101B).
2. Executes an action to select a record on that form.
3. Executes another action on the P01012_W01012A form and updates the Name field.
4. Executes another action to press the OK button.
5. Executes another action to press the Close button on W01012A to close the form.
6. Closes the stack.

Example - Application Stack

```
public void appStack() throws Exception
{

loginEnv.getUsedCapabilities().add("applicationStack");
ApplicationStack appStackAddress = new ApplicationStack();
FormRequest formRequest = new FormRequest(loginEnv);
formRequest.setReturnControlIDs("1");
formRequest.setFormName("P01012_W01012B");

formRequest.setReturnControlIDs("54|1[19,20]");
formRequest.setFormServiceAction("R");
formRequest.setMaxPageSize("5");
FSREvent findFSREvent = new FSREvent();

findFSREvent.setFieldValue("54", "E");
findFSREvent.doControlAction("15"); // Find button
formRequest.addFSREvent(findFSREvent);


ObjectWriter writer = loginEnv.getObjectMapper().writerWithDefaultPrettyPrinter();
out.println(writer.writeValueAsString(formRequest));

//open P01012_W01012B
String response = appStackAddress.open(loginEnv, formRequest);
out.println(writer.writeValueAsString(loginEnv.getObjectMapper().readTree(response)));

//check if in find browse
if (appStackAddress.getLastAppStackResponse().checkSuccess("P01012_W01012B"))
{
//select a record
ActionRequest actionRequest = new ActionRequest();
actionRequest.setReturnControlIDs("28"); //the form changes these return control IDs are
 for the next form
actionRequest.setFormOID("W01012B");
FSREvent selectFSREvent = new FSREvent();
selectFSREvent.selectRow("1", 3);
selectFSREvent.doControlAction("14"); //select button
actionRequest.addFSREvent(selectFSREvent);

response = appStackAddress.executeActions(loginEnv, actionRequest);
out.println(writer.writeValueAsString(loginEnv.getObjectMapper().readTree(response)));

//check if in fix inspect
if (appStackAddress.getLastAppStackResponse().checkSuccess("P01012_W01012A"))
```

```
{
//Change name - now on form A
ActionRequest actionRequestName = new ActionRequest();
actionRequestName.setReturnControlIDs("54|1[19,20]"); //form is going to change again
 these are for the next form
actionRequestName.setFormOID("W01012A");
FSREvent updateFSREvent = new FSREvent();

updateFSREvent.setFieldValue("28", "AIS APP Stack TEST"); //change name field
updateFSREvent.doControlAction("11"); //ok

actionRequestName.addFSREvent(updateFSREvent);

response = appStackAddress.executeActions(loginEnv, actionRequestName);
out.println(writer.writeValueAsString(loginEnv.getObjectMapper().readTree(response)));

 //IMPORTANT: here you would have to de-serialize the response to check if there were
 errors on the form after pressing okay, if so you could continue to close the A form and
 go back to the B form
if (appStackAddress.getLastAppStackResponse().checkSuccess("P01012_W01012A"))
{
//press find again (to see name change) then close the stack
ActionRequest actionRequestClose = new ActionRequest();
FSREvent closeFSREvent = new FSREvent();
actionRequestClose.setReturnControlIDs("54|1[19,20]"); //form is changing these are the
 controls of the returned form
actionRequestClose.setFormOID("W01012A");
closeFSREvent.doControlAction("12"); //close
actionRequestClose.addFSREvent(closeFSREvent);

response = appStackAddress.close(loginEnv, actionRequestClose);
out.println(writer.writeValueAsString(loginEnv.getObjectMapper().readTree(response)));
}
}

}

}
```

Media Object Operations

Media objects in EnterpriseOne store file attachments and text attachments. The media object operations in the AIS Client Java API use the following items to identify individual media object attachments for a record:

- Media object name, for example `GT00202`.
- Media object key to identify the record. This is a bar delimited key string, for example `6540|3|1`.
- Sequence number to identify the individual attachment for a record.

Get Text

This operation returns the text in the first text attachment, as shown in the code in *Example - Media Object Get Text Operation*:

Example - Media Object Get Text Operation

```
try{
MediaObjectGetTextRequest moGetText = new MediaObjectGetTextRequest(loginEnv);

moGetText.setFormName("P01012_W01012B");
moGetText.setVersion("ZJDE0001");
moGetText.setMoStructure("ABGT");

//set mo key - in this case it's just AB number
moGetText.addMoKeyValue("7");

MediaObjectGetTextResponse response = MediaObjectOperations.getTextMediaObject(loginEnv,
 moGetText);

System.out.println(response.getText());
}
catch (Exception e){
//handle exception
}
```

Update Text

This operation updates (replaces or appends to) the first text media object, as shown in the code in *Example - Media Object Update Text Operation*:

Example - Media Object Update Text Operation

```
try{
MediaObjectUpdateTextRequest moSetText = new MediaObjectUpdateTextRequest(loginEnv);
moSetText.setFormName("P01012_W01012B");
moSetText.setVersion("ZJDE0001");
moSetText.setMoStructure("ABGT");

//set mo key
moSetText.addMoKeyValue("7");
moSetText.setAppendText(true);
//set text
moSetText.setInputText("Append This text");

MediaObjectUpdateTextResponse response =
 MediaObjectOperations.updateTextMediaObject(loginEnv, moSetText);

System.out.println("Status " + response.getUpdateTextStatus());
}
catch (Exception e){
//handle exception
}
```

List

MediaObjectListRequest is the input to the media object getMediaObjectList operation. The following table describes the attributes in the request that control the list that is returned:

Field	Type	Description
includeURLs	boolean	Valid values are: • True • False When true, includes the URL for downloading the media object, which can be used in conjunction with a download request at a later time. This only applies to the media object file type.
includeData	boolean	Valid values are: • True • False When true, if the file is an image, it includes the base64 encoded data for a thumbnail sized image.
moTypes	String	Use a constant defined in MediaObjectListRequestValid, which includes these constants: • MediaObjectListRequest.MO_TYPE_TEXT • MediaObjectListRequest.MO_TYPE_FILE • MediaObjectListRequest.MO_TYPE_QUEUE • MediaObjectListRequest.MO_TYPE_URL
extensions	String	File extensions to include in the response, which enables you to filter out undesired extensions.
thumbnailSize	<String> int	Size of the thumbnail image returned as base64 data.

The code in *Example - Saving Thumbnail Images for Image Media Object Attachments* is an example of saving the set of thumbnail images for image media object attachments. It includes specified extensions for the first file type attachments. The includeData value is set to include the thumbnail data. If the file is not a PDF (a non-image type), the thumbnail data is saved to a local file.

Example - Saving Thumbnail Images for Image Media Object Attachments

```
import Java.awt.image.BufferedImage;
import sun.misc.BASE64Decoder;

public void listMediaObject() throws Exception
{
final String MO_STRUCTURE = "ABGT";
final String MO_APP = "P01012_W01012B";
final String MO_VERSION = "ZJDE0001";
final String MO_KEY = "479";
final int MO_THUMBSIZE = 50;
final String FILE_LOCATION = "C:\\temp\\AISClientDownloads\\";

//set request info include URLs so they don't have to be fetched later
MediaObjectListRequest mediaObjectListRequest = new MediaObjectListRequest(loginEnv);
mediaObjectListRequest.setFormName(MO_APP);
```

```
mediaObjectListRequest.setVersion(MO_VERSION);
mediaObjectListRequest.setIncludeURLs(false);
mediaObjectListRequest.setIncludeData(true);
mediaObjectListRequest.setMoStructure(MO_STRUCTURE);
mediaObjectListRequest.setThumbnailSize(MO_THUMBSIZE); //available in tools 9.1.5+ only

//set the moKey
mediaObjectListRequest.addMoKeyValue(MO_KEY);

// - Date Example, if MO key includes a date value -
//mediaObjectListRequest.addMoKeyValue(AISClientUtilities.convertMillisecondsToYMDString(mydate.getTime()));

//I only want
filesmediaObjectListRequest.addMoType(mediaObjectListRequest.MO_TYPE_FILE);
mediaObjectListRequest.addMoType(mediaObjectListRequest.MO_TYPE_QUEUE);


//I only want these types
mediaObjectListRequest.addExtension("jpg");
mediaObjectListRequest.addExtension("gif");
mediaObjectListRequest.addExtension("jpeg");
mediaObjectListRequest.addExtension("pdf");

//get the list of available files for this media object
MediaObjectListResponse mediaObjectListResponse = MediaObjectOperations.getMediaObjectList(loginEnv,
 mediaObjectListRequest);

if (mediaObjectListResponse != null)
{

for (int i = 0; i < mediaObjectListResponse.getMediaObjects().length; i++)
{

FileAttachment fileAt = new FileAttachment();
fileAt.setThumbFileLocation(mediaObjectListResponse.getMediaObjects()[i].getThumbFileLocation());
fileAt.setItemName(mediaObjectListResponse.getMediaObjects()[i].getItemName());
fileAt.setFileName(mediaObjectListResponse.getMediaObjects()[i].getFile());
fileAt.setDownloadUrl(mediaObjectListResponse.getMediaObjects()[i].getDownloadUrl());
fileAt.setSequence(mediaObjectListResponse.getMediaObjects()[i].getSequence());

//if it's an image, save the thumnail data to a file
if (!fileAt.getFileName().contains("pdf"))
{
BufferedImage image = decodeToImage(mediaObjectListResponse.getMediaObjects()[i].getData());
if (image != null)
{
File file = new File(fileAt.getFileName());

File outputfile = new File(FILE_LOCATION + "thumb_" + file.getName());
ImageIO.write(image, "jpg", outputfile);
}

}
}


}
}
    public static BufferedImage decodeToImage(String imageString)
    {

        BufferedImage image = null;
        byte[] imageByte;
        try
        {
            BASE64Decoder decoder = new BASE64Decoder();
            imageByte = decoder.decodeBuffer(imageString);
            ByteArrayInputStream bis = new ByteArrayInputStream(imageByte);
            image = ImageIO.read(bis);
            bis.close();
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
```

```
        return image;
    }
```

Upload

To upload a file, you need to provide the media object data structure key information:

- A string with the location of the local file to be uploaded.
- A name for the item. If you do not supply a name, the file name is used.

The code in *Example - Media Object Upload* uploads a file to the Address Book media object for address book number 479. The response to the upload request will print the name and sequence number of the new record.

Example - Media Object Upload

```
public void uploadFile(String fileLocation, String itemName) throws Exception
{

final String MO_STRUCTURE = "ABGT";
final String MO_APP = "P01012_W01012B";
final String MO_VERSION = "ZJDE0001";
final String MO_KEY = "479";

MediaObjectUploadRequest mediaObjectUploadRequest = new
 MediaObjectUploadRequest(loginEnv);
mediaObjectUploadRequest.setFormName(MO_APP);
mediaObjectUploadRequest.setVersion(MO_VERSION);
mediaObjectUploadRequest.setMoStructure(MO_STRUCTURE);

//set the moKey
mediaObjectUploadRequest.addMoKeyValue(MO_KEY);

String fileLocation = "C:\\temp\\images\\IMG_20001.jpg";
String itemName = "Joe's Photo";
FileAttachment newFileAttachment = new FileAttachment();
newFileAttachment.setFileLocation(fileLocation);
newFileAttachment.setItemName(itemName);

//set the file to the new one they just saved
mediaObjectUploadRequest.setFile(newFileAttachment);

//Upload to Server
MediaObjectUploadResponse response = MediaObjectOperations.uploadMediaObject(loginEnv,
 mediaObjectUploadRequest);

out.println("NEW MO: " + response.getItemName());
out.println("NEW MO SEQ: " + response.getSequence());

}
```

Download

To download a file, you can provide the following input:

- **downloadURL**. (String) (Optional) If you requested this value from the list request, send it to the server and it will save the step of fetching this URL. If you do not pass a value, the URL will be fetched by AIS.
- **sequence**. (int) (Required) The sequence number of the attachment for this media object record.
- **height**. (int) (Optional) If the file you are downloading is an image, the AIS Server will scale the image to the requested height.
- **width**. (int) (Optional) If the file you are downloading is an image, the AIS Server will scale the image to the requested width.
- **fileName**. (String) (Required) Provide a name for the downloaded file, if desired you can use the same name returned in the list response.

The code in *Example - Media Object Download* is an example of downloading a media object attachment. Executing the *getMediaObjectList* operation produces a FileAttachment object that contains the sequence and media object file name. It passes the FileAttachment object into this method where a call is made to *downloadMediaObject* operation (passing a desired file location). The response will include the location of the saved file.

Example - Media Object Download

```
public void downloadFile(FileAttachment fileAt) throws Exception
{

final String MO_STRUCTURE = "ABGT";
final String MO_APP = "P01012_W01012B";
final String MO_VERSION = "ZJDE0001";
final String MO_KEY = "479";
final String FILE_LOCATION = "C:\\temp\\AISClientDownloads\\";

//set the download request info - don't need mo key because we have the list already
MediaObjectDownloadRequest mediaObjecDownloadRequest = new
 MediaObjectDownloadRequest(loginEnv);

mediaObjecDownloadRequest.setFormName(MO_APP);
mediaObjecDownloadRequest.setVersion(MO_VERSION);
mediaObjecDownloadRequest.setMoStructure(MO_STRUCTURE);
mediaObjecDownloadRequest.setWidth(700);
mediaObjecDownloadRequest.addMoKeyValue(MO_KEY);
mediaObjecDownloadRequest.setSequence(fileAt.getSequence());
mediaObjecDownloadRequest.setFileName(fileAt.getFileName());

// download the file and save to file location
MediaObjectDownloadResponse mediaObjecDownloadResponse =
 MediaObjectOperations.downloadMediaObject(loginEnv, mediaObjecDownloadRequest,
 FILE_LOCATION);

out.println("Downloaded File: " + mediaObjecDownloadResponse.getFile().getFileLocation());


}
```

Add URL (API 1.0)

To add a URL type media object, provide the media object keys as well as the URL text, such as:

`http://www.domainname.com.`

The code in *Example - Adding a URL Media Object* is an example of adding a URL type media object:

Example - Adding a URL Media Object

```
public void addURL() throws Exception {

   final String MO_STRUCTURE = "ABGT";
   final String MO_APP = "P01012_W01012B";
   final String MO_VERSION = "ZJDE0001";
   final String MO_KEY = "479";
   final String URL_TEXT = "http://www.google.com";


   //set request info include URLs so they don't have to be fetched later
   MediaObjectAddUrlRequest mediaObjectAddUrlRequest = new
 MediaObjectAddUrlRequest(loginEnv);

   mediaObjectAddUrlRequest.setFormName(MO_APP);
   mediaObjectAddUrlRequest.setVersion(MO_VERSION);
   mediaObjectAddUrlRequest.setMoStructure(MO_STRUCTURE);

   mediaObjectAddUrlRequest.addMoKeyValue(MO_KEY);
   mediaObjectAddUrlRequest.setUrlText(URL_TEXT);

   MediaObjectAddUrlResponse mediaObjectAddUrlResponse = new MediaObjectAddUrlResponse();

   mediaObjectAddUrlResponse = MediaObjectOperations.addUrlMediaObject(loginEnv,
 mediaObjectAddUrlRequest);

   System.out.println("Saved URL: " + mediaObjectAddUrlResponse.getSaveURL());
   System.out.println("Sequence: " + mediaObjectAddUrlResponse.getSequence());

}
```

Delete

To delete a media object file, provide the media object keys and the individual sequence of the attachment you want to delete.

The code in *Example - Deleting a Media Object* is an example of deleting a media object file. This example assumes the FileAttachment object has already been created. It uses the sequence and location values from that object to request the delete operation.

Example - Deleting a Media Object

```
public void deleteFile(FileAttachment fileAt) throws Exception
{
```

```
MediaObjectDeleteRequest mediaObjectDelete = new MediaObjectDeleteRequest(loginEnv);

//set request info
mediaObjectDelete.setFormName(MO_APP);
mediaObjectDelete.setVersion(MO_VERSION);
mediaObjectDelete.setMoStructure(MO_STRUCTURE);

//set mo key
mediaObjectDelete.addMoKeyValue(MO_KEY);
mediaObjectDelete.setSequence(fileAt.getSequence());
mediaObjectDelete.setFileLocation(fileAt.getFileLocation());

//call delete operation to remove from E1 server and remove from local file system
MediaObjectDeleteResponse response = MediaObjectOperations.deleteMediaObject(loginEnv,
 mediaObjectDelete);

System.out.println("MO Delete Response Status  " + response.getDeleteStatus());


}
```

Processing Option Service

The AIS Server provides a processing option service that enables you to retrieve the processing option fields and values for an application and version in EnterpriseOne.

The key strings can be derived by creating a type definition on the PO Data Structure in Object Management Workbench (OMW). The italicized portion of the #define below shows the key string for the example.

```
#define IDERRmnNetQuebecTaxCredit_27              27L
```

There are six supported data types. These are based on the data item used in the Processing Option Design Aid for each option.

You can get the type of the option before attempting to cast it, which is the recommended method. Or you can just cast it to the type you expect, because it is unlikely to change. The default is String, so you will always be able to get to a string version of the option value.

Type Code	Type Constant	Java Type	JDE DD Type
1	STRING_TYPE	String	String
2	CHAR_TYPE	String	Character
9	BIG_DECIMAL_TYPE	BIG Decimal	Math Numeric
11	DATE_TYPE	Date	Date
15	INTEGER_TYPE	Integer	Integer

Type Code	Type Constant	Java Type	JDE DD Type
55	CALENDAR_TYPE	Calendar	Utime

The code in *Example - Retrieving Processing Options with the Processing Option Service* is an example of retrieving processing options for P0801, version ZJDE0001. First it populates the request values and calls the poRequest service. After deserializing the response to a ProcessingOptionSet object, it uses the getOptionValue method to retrieve the value for a specific processing option based on the key string.

Example - Retrieving Processing Options with the Processing Option Service

```
public void processingOption()throws Exception{

//add capability to used (or add during login to required)
loginEnv.getUsedCapabilities().add("processingOption");
ProcessingOptionRequest poRequest = new ProcessingOptionRequest(loginEnv);
poRequest.setApplicationName("P0801"); //application
poRequest.setVersion("ZJDE0001"); //version

String response =
JDERestServiceProvider.jdeRestServiceCall(loginEnv, poRequest,
 JDERestServiceProvider.POST_METHOD, JDERestServiceProvider.PO_SERVICE);


//response can be serialized to ProcessingOptionSet class
ProcessingOptionsSet poSet = loginEnv.getObjectMapper().readValue(response,
 ProcessingOptionsSet.class);

//get the value for quebec tax credit using key string
BigDecimal quebecTaxCred = (BigDecimal)poSet.getOptionValue("mnNetQuebecTaxCredit_27");

System.out.println("mnNetQuebecTaxCredit_27 value: " + quebecTaxCred);
}
```

Task Authorization Service

The task authorization service enables you to retrieve the authorized tasks in a specific EnterpriseOne task view or under a specific task within a task view.

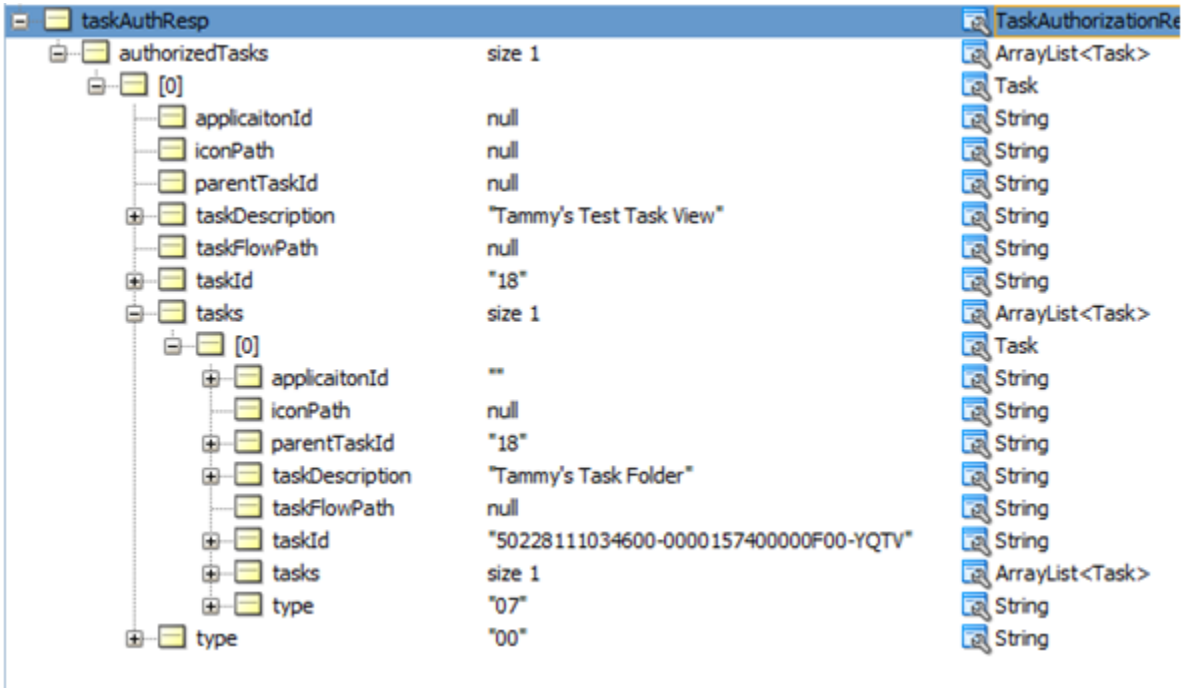
The code in *Example - Retrieving Tasks with the Task Authorization Service* is an example of retrieving the tasks under task view 18.

Example - Retrieving Tasks with the Task Authorization Service

```
public void taskAuthorization() throws Exception
{

String taskViewId = "18";
```

```
loginEnv.getRequiredCapabilities().add("taskAuthorization");
TaskAuthorizationRequest taksAuthReq = new TaskAuthorizationRequest(loginEnv);
taksAuthReq.setTaskViewId(taskViewId);


String response = JDERestServiceProvider.jdeRestServiceCall(loginEnv, taksAuthReq,
 JDERestServiceProvider.POST_METHOD, JDERestServiceProvider.TASK_AUTHORIZATION);


//response can be serialized to TaskAuthorizationResponse class
TaskAuthorizationResponse taskAuthResp = loginEnv.getObjectMapper().readValue(response,
 TaskAuthorizationResponse.class);
System.out.println(writer.writeValueAsString(taskAuthResp));

    }
```

The TaskAuthorizationResponse object contains and array of Task type object, each with its own array of Task type objects. Although the structure supports an "infinite" number of levels, the service returns only two levels below the top task view or menu requested. You may call the service again to drill down two more levels, and so on.



To drill down another two levels, pass values in for taskViewId, taskId, and parentTaskId (which you received from the original request). The results will include children two levels down from the taskViewId passed.

Note:

- *JD Edwards EnterpriseOne Tools Solution Explorer Guide* for information about tasks and task views in EnterpriseOne.

Logging Service

The logging service enables the AIS client to log a message in the AIS Server log. The code in *Example - Logging Service Code* is an example of using the logging service.

Example - Logging Service Code

```
//do this once and it will be stored in the login environment to be used over and over,
 values are optional, they will just show as null in the log if you don't set them
loginEnv.setApplicationName("My Client Application");
loginEnv.setApplicationVersion("Client Version");

//do this every time you want to send a log to AIS
AISClientLogger.log(loginEnv, "Warn Log sent from Client to AIS
 Server",AISClientLogger.WARN);
```

In this example, the log entry in the AIS Server log would be:

```
AIS LOG REQUEST: --Level 2 --Application: My Client Application --Application Version:
 Client Version --User: jde --Device Name: javaclient --Log Message: Warn Log sent from
 Client to AIS Server
```

Query Support

The AIS Client Java API supports sending an ad hoc query to EnterpriseOne. Starting with EnterpriseOne release 9.2.0.2 and AIS Client Java API 1.3.1, it also supports invoking an existing query. You can configure an AIS client to perform one or the other, not both.

You can include an ad hoc query in a form service request. You can include an invocation of an existing query in a form service request or data service request.

This section contains the following topics:

- *Adding an Ad Hoc Query*
- *Adding a Request to an Existing (Saved) Query (API 1.3.1 and EnterpriseOne Tools 9.2.0.2)*
- *Additional Query Capabilities (API 1.4.2 and EnterpriseOne Tools 9.2.1)*

Adding an Ad Hoc Query

You can configure a form service request to send ad hoc queries to EnterpriseOne web client application forms that support the query control.

To add a query, you include a single query object in the form service request. A query object includes parameters that contain the same query criteria that you would use to set up a query in EnterpriseOne. The parameters determine:

- How the query runs.

  You can configure query option parameters to load grid records in the form or clear all other fields in the form before the query runs. You can also specify whether the results of the query should match all (AND) or any (OR) of the conditions specified in the query.

- The conditions of the query.

  The query object includes condition parameters that specify the control ID of the columns or fields that you want to query and an operator for filtering results that are equal to, greater than, or less than a particular value.

  > **Note:** Queries will work only if the field or columns identified in the query are part of the business view.

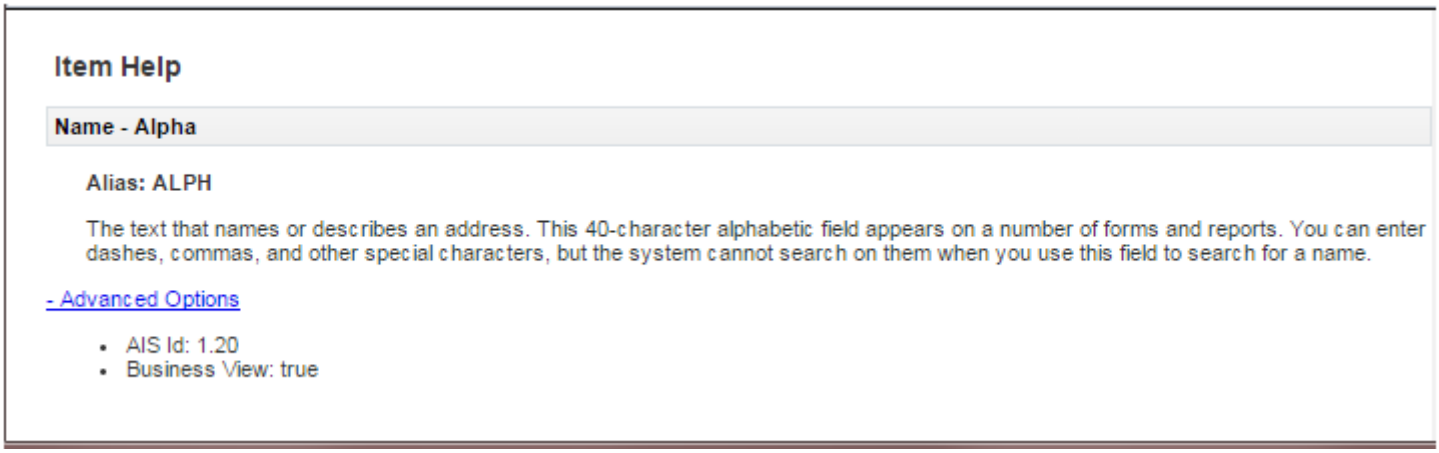
- The value used for the search criteria in the query.

  The query object includes value parameters that specify the value or range of values that you want displayed in the query results.

Before you add a query object to a form service request, access the form in the EnterpriseOne web client and use the query control to gather the criteria for the query object parameters. For more information about setting up a query, see *"Understanding the Query Control" in the JD Edwards EnterpriseOne Tools Using and Approving User Defined Objects Guide* .

Also, in the EnterpriseOne form, you need to identify the control ID of the field or column that you want to query, and verify that the field or column is part of the business view. To do so, click the Help button (question mark in the upper right corner of a form) and then click the Item Help option to access field-level help. With the field level help activated, you can click in a field or column to access the control ID and business view information, which is displayed under the Advanced Options section as shown in *Example - Example of Control ID and Business View Information Displayed under Advanced Options in the EnterpriseOne Web Client Item Help*.

Example - Example of Control ID and Business View Information Displayed under Advanced Options in the EnterpriseOne Web Client Item Help



In the Item Help, the syntax of the control ID is 1.20 with 1 representing the grid ID and 20 representing the column ID, which are separated by a dot (.). In the parameter for the query request, the same control ID must be presented with the following syntax: 1[20]. See *Table 4: Query Condition Parameters* for more information.

Query Object Parameters

The following tables provide descriptions of the option, condition, and value parameters for a query object.

Query Option Parameters

Parameter	Description	Values
autoFind	Directs the query to automatically press Find on the form to populate the grid records. You do not need to include events to press the Find button if you use autoFind.	true, false
matchType	Determines if you want the query to search for records that match all (AND) or any (OR) of the specified conditions.	MATCH_ALL, MATCH_ANY
autoClear	Determines if you want to clear all other fields on the form (for example default filter fields).	true, false

Query Condition Parameters

Parameter	Description	Value
controlId	The control ID that the condition applies to. This is the field that you add to the query from the form when using the web client to create a Query. It is either a filter field or a grid column that is associated with the business view.	Example of control IDs: "28", "1[34]"
operator	The comparison operation to use with the query.	For all types, valid values are: BETWEEN, LIST, EQUAL, NOT_EQUAL, LESS, LESS_EQUAL, GREATER, GREATER_EQUAL For strings, valid values are: STR_START_WITH, STR_END_WITH, STR_CONTAIN, STR_BLANK, STR_NOT_BLANK

Query Value Parameters

Parameter	Description	Value
content	This is either a literal value to be used in the comparison operation, or it relates to a special value ID.	Examples of values are: "23", "Joe", "2"

Parameter	Description	Value
specialValueId	This is a special value, mostly for dates that might be the current day (TODAY), or calculated dates from the current day. For calculated dates, the content field is used in the calculation.	Valid values are: LITERAL, TODAY, TODAY_PLUS_DAY, TODAY_MINUS_DAY, TODAY_PLUS_MONTH, TODAY_MINUS_MONTH, TODAY_PLUS_YEAR, TODAY_MINUS_YEAR

Example - Query - Java API

The sample code in this example shows a query executed in the W42101C form. This query attempts to match the following specified conditions:

- Line Number equal to 2.
- Requested Date within the last 2 years.
- Sold To between 7000 and 8000.
- Company is one of the values in the list "00070,00077".

The response will contain the JSON for the form with the matching records in the grid.

```
public void queryP42101() throws Exception
   {

        loginEnv.getUsedCapabilities().add("query");
        FormRequest formRequest = new FormRequest(loginEnv);
        formRequest.setFormName("P42101_W42101C");
        formRequest.setReturnControlIDs("350|360|41[129,130,116,125]");
        formRequest.setFormServiceAction(formRequest.ACTION_READ);
        formRequest.setFindOnEntry("TRUE");
        formRequest.setMaxPageSize("20");
        Query query = new Query(loginEnv);


        //auto find
        query.setAutoFind(true);

        //match all
        query.setMatchType(Query.MATCH_ALL);

        //clear any defaulted filters
        query.setAutoClear(false);

        //line number equals 2
        NumberCondition condN = query.addNumberCondition("41[129]",
NumericOperator.EQUAL());
        condN.setValue(2);

        //Requested Date within two years from today
        DateCondition condD = query.addDateCondition("41[116]", DateOperator.GREATER());
        condD.setSpecialDateValue(DateSpecialValue.TODAY_MINUS_YEAR(), 2);

        //Sold To 125
        BetweenCondition condST = query.addBetweenCondition("41[125]");
        condST.setValues("7000", "8000");
```

```
        //company in list
        ListCondition list1 = query.addListCondition("360");
        list1.addValue("00070");
        list1.addValue("00077");

        //set it in the request
        formRequest.setQuery(query);

        ObjectWriter writer = loginEnv.getObjectMapper().writerWithDefaultPrettyPrinter();
        out.println(writer.writeValueAsString(formRequest));


        String response = JDERestServiceProvider.jdeRestServiceCall(loginEnv, formRequest,
 JDERestServiceProvider.POST_METHOD, JDERestServiceProvider.FORM_SERVICE_URI);


}
```

Adding a Request to an Existing (Saved) Query (API 1.3.1 and EnterpriseOne Tools 9.2.0.2)

The AIS Client Java API includes a QueryObjectName parameter in both the FormServiceRequest and DataServiceRequest objects. You can include a query object name in this parameter to invoke an existing query in EnterpriseOne through a form service request or a data service request.

You can locate the query object name in the EnterpriseOne web client.

> **CAUTION:** In a data service request, do NOT use the queryObjectName parameter if the DataService type is a COUNT. The result will not be accurate because a saved query cannot be applied to a COUNT type data service request.

Additional Query Capabilities (API 1.4.2 and EnterpriseOne Tools 9.2.1)

This sections describes the following services for working with saved queries and ad hoc queries:

- *List Available Queries*
- *Get Query Details*
- *Complex Query*
- *Application Query In Data*
- *Query Combining*
- *Query with Aggregation*

List Available Queries

The "list available queries" service retrieves a list of available queries for an application for the current user. *Example - LIst Available Queries* is an example of the Java code. *Example - LIst Available Queries* is an example of the response in JSON.

Example - LIst Available Queries

```
//create the request
ListAvailableQueriesRequest queriesRequest = new ListAvailableQueriesRequest(loginEnv);
//Pass in the application name and form name.
queriesRequest.setAppName("P01012");
queriesRequest.setFormName("W01012B");
//excute the call
ListAvailableQueriesResponse responseObjects = queriesRequest.execute();
//Response contains a list of UDO objects grouped by their UDO status
//Get the omwObjectName for the first (0) shared (4) query object
String omwObjectName =
 responseObjects.getUdoObjects().get(4).getItems().get(0).getOmwObjectName()
```

Example - List Available Queries JSON Response

```
{
  "managerTitle" : "Query Manager",
  "udoObjects" : [ {
    "group" : "Personal",
    "items" : [ {
      "name" : "6001",
      "omwObjectName" : "QRY01012B_1607180001CUST",
      "user" : "JDE",
      "description" : "6001",
      "app" : "P01012",
      "form" : "W01012B",
      "version" : "ZJDE0001"
    }]
  }, {
    "group" : "Pending Approval",
    "items" : [{
      "name" : "Simple EE Search",
      "omwObjectName" : "QRY01012B_1606300001CUST",
      "user" : "JDE",
      "description" : "Simple EE Search",
      "app" : "P01012",
      "form" : "W01012B",
      "version" : "ZJDE0001"
    } ]
  }, {
    "group" : "Rework"
  }, {
    "group" : "Reserved"
  }, {
    "group" : "Shared",
    "items" : [ {
      "name" : "All",
      "omwObjectName" : "QRY01012B_1602160001CUST",
      "user" : "*PUBLIC",
      "description" : "All in the name",
      "app" : "P01012",
      "form" : "W01012B",
```

```
      "version" : "ZJDE0001"
    }]
  } ],
  "activeObject" : {
    "name" : "C",
    "omwObjectName" : "QRY01012B_1608240001CUST",
    "user" : "JDE",
    "description" : "C"
  }
}
```

Get Query Details

The "get query details" service retrieves the details of an individual query by passing the omwObjectName. *Example - Get Query Details* is an example of the Java code. *Example - Get Query Details JSON Response* is an example of the response in JSON.

Example - Get Query Details

```
//create the request
GetQueryByKeyRequest getindividual = new GetQueryByKeyRequest(loginEnv);
//set the OMW object name getindividual.setOmwObjectName("QRY01012B_1603150008CUST");
//execute the request
GetQueryByKeyResponse resp = getindividual.execute();
```

Example - Get Query Details JSON Response

```
{
  "activeObject" : {
    "name" : "6001",
    "omwObjectName" : "QRY01012B_1603150008CUST",
    "user" : "*PUBLIC",
    "description" : "6001",
    "metaData" : {
      "bsvw" : "V0101E",
      "andQuery" : true,
      "autoClear" : false,
      "autoFind" : true,
      "conditions" : [ {
        "leftServerId" : "qbe0_1.19",
        "leftId" : "qbe0_1.0",
        "ddAlias" : "AN8",
        "table" : "F0101",
        "dataType" : 0,
        "display" : "Address Number (QBE)",
        "rightWidth" : 84,
        "maxLength" : 8,
        "operatorId" : 5,
        "rightOperand" : [ {
          "value" : "6001",
          "svId" : 0
        } ]
      } ]
    }
  }
}
```

Complex Query

The complex query service provides the capability to combine multiple ad hoc query objects with AND/OR relationships in a single service request. Use this service to add more complex filtering than is available in a single query object.

Example - Complex Query shows a complex query that will find and return records that contain the words "red" AND "bike" in any of the four specified columns.

Example - Complex Query

```
DataRequest f4101 = new DataRequest(loginEnv);
f4101.setDataServiceType(DataRequest.TYPE_BROWSE);
f4101.setTargetName("F4101");
f4101.setTargetType(DataRequest.TARGET_TABLE);
f4101.setFindOnEntry(FormRequest.TRUE);
        f4101.setReturnControlIDs("F4101.ITM|F4101.DSC1|F4101.DSC2|F4101.SRTX|
F4101.LITM");

f4101.setOutputType(loginEnv, DataRequest.GRID_DATA_OUTPUT_TYPE);

f4101.addOrderBy(loginEnv, "F4101", "ITM", OrderByDirection.ORDER_DIRECT_ASCENDING());

Query red = new Query(loginEnv);
red.setMatchType(Query.MATCH_ANY);
red.addStringCondition("F4101.DSC1", StringOperator.CONTAINS(), "red");
red.addStringCondition("F4101.DSC2", StringOperator.CONTAINS(), "red");
red.addStringCondition("F4101.SRTX", StringOperator.CONTAINS(), "red");
red.addStringCondition("F4101.LITM", StringOperator.CONTAINS(), "red");

Query bike = new Query(loginEnv);
bike.setMatchType(Query.MATCH_ANY);
bike.addStringCondition("F4101.DSC1", StringOperator.CONTAINS(), "bike");
bike.addStringCondition("F4101.DSC2", StringOperator.CONTAINS(), "bike");
bike.addStringCondition("F4101.SRTX", StringOperator.CONTAINS(), "bike");
bike.addStringCondition("F4101.LITM", StringOperator.CONTAINS(), "bike");

ComplexQuery complexQuery = new ComplexQuery();
complexQuery.setAutoFind(true);
complexQuery.addQueryAnd(red);
//put an AND operator between the two conditions
complexQuery.addQueryAnd(bike);

f4101.setQuery(complexQuery);


String response = JDERestServiceProvider.jdeRestServiceCall(loginEnv, f4101,
 JDERestServiceProvider.POST_METHOD, JDERestServiceProvider.DATA_SERVICE_URI);
```

Application Query In Data

The "application query in data" service provides the capability to invoke a saved query within a data service call. The "application query in data" service can invoke a saved query associated to an application form.

The data service provides data query or count requests over tables or business views. However, users most often build their queries in the application form, not in the EnterpriseOne Data Browser. Application forms use the same business views that are available to data service calls. Therefore, in the "application query in data" service, you can specify the business view associated with the form that contains the saved queries. This enables you to call a saved query in a form when calling a data service over that same business view.

To identify the business view associated with a form that contains the saved queries, you can run a form service request in demo mode using the setFormServiceDemo(true) method. The response displays the business view name, for example:

```
"bsvwName" : "V060116B",
```

You can use the available query service to identify the queries defined for that form and their OMW object IDs. You can then pass the query ID into the queryObjectName parameter of the data request, for example:

```
dataRequest.setQueryObjectName(loginEnv, "QRY0801A_1606010003CUST");
```

The query associated with the application can then be applied to the data request call.

Query Combining

The query combining service provides the capability to include both a saved query and an ad hoc query in form service and data service requests. Query combining enables you to apply both sets of filters, essentially putting an AND operation between them. This can be useful if you want to include a generic query that you always want to apply (the saved query) with an ad hoc query that can change on each request to filter the results even further.

Query with Aggregation

The "query with aggregation" service provides the capability to include a saved query, ad hoc query, or both in a data aggregation request. This enables you to reduce the records included in any aggregation. If you include both, the saved query and ad hoc query are applied using an AND operation.

Jargon Service

The jargon service enables you to retrieve data item descriptions for any EnterpriseOne data dictionary item based on the users language and jargon (system) code. This service depends on language packs applied to the EnterpriseOne system as well as data item description overrides entered with jargon codes. If there is no language pack or overrides, the base data item description is returned.

The capability name for the jargon service is "jargon". The AIS Server must have this capability to be able to process jargon service requests.

Example - Jargon Service Java API

In this example, several data items are loaded into the JargonRequest object, the service is called, and the descriptions in the response are printed out.

```
public void jargonService() throws Exception
    {
    //this uses the jargon capability
        loginEnv.getRequiredCapabilities().add("jargon");

  //create the request object, seeding it with a default system code of 01
        JargonRequest jargonRequest = new JargonRequest(loginEnv, "01"); // with default
 system code

   //fill the list in the request with data items
    jargonRequest.addDataItem("AN8");  //uses default system code
```

```
    jargonRequest.addDataItem("MCU","04");  //use system code 04 for this one
    jargonRequest.addDataItem("PAN8");
    jargonRequest.addDataItem("ITM","55");


        //call the jargon service
        String response = JDERestServiceProvider.jdeRestServiceCall(loginEnv,
jargonRequest, JDERestServiceProvider.POST_METHOD,
JDERestServiceProvider.JARGON_SERVICE);


        //response can be serialized to JargonResponse class
        JargonResponse jargonResponse = loginEnv.getObjectMapper().readValue(response,
JargonResponse.class);


        //print the response
        if(jargonResponse != null)
        {
            if(jargonResponse.getRequestedItems() != null &&
jargonResponse.getRequestedItems().size() >0 )
            {
                for(JargonResponseItem item: jargonResponse.getRequestedItems())
                {
                    System.out.println("Item " + item.getSzDict() + " " +
item.getRowDescription());
                }
            }
        }
    }
```

Data Service (API 1.1.0)

This section contains the following topics:

- *Understanding the Data Service*
- *Determining the Maximum Records Returned in a Data Service*
- *Data Service Data Aggregation (API 1.3.1 and EnterpriseOne Tools 9.2.0.2)*
- *Data Service Data Aggregation - Currency Decimals Support (API 1.3.3 and EnterpriseOne Tools 9.2.0.3)*

Understanding the Data Service

The AIS Server provides an endpoint called "dataservice" for data query or count requests over tables or business views.

Data service calls are made using the DataRequest object. If you use the data service, you must include the "dataservice" capability in the required or used capabilities list.

Data Service Request Required Parameters

Parameter	Description	Values
targetName	The name of the table or view to count or query.	Example values: F0101 or V4210A

Parameter	Description	Values
targetType	The object type to count or query: table or business view.	DataRequest.TARGET_TABLE DataRequest.TARGET_VIEW
dataServiceType	The type of operation to be performed: count or query (represented by the value BROWSE).	DataRequest.TYPE_COUNT DataRequest.TYPE_BROWSE

Data Service Request Optional Parameters

Parameter	Description	Values
findOnEntry	This parameter determines if the service performs an automatic find.	FormRequest.TRUE FormRequest.FALSE
returnControlIDs	The columns of the table or business view to be returned in a query response (pipe delimited).	Example values: F0101.AN8|F0101.PA8|F0101.ALPH
query	A query object which is built using column IDs for the control IDs.	
OrderBy (Available starting with EnterpriseOne Tools 9.2.0.5 and API 1.4.0)	For calls to an EnterpriseOne Find/Browse form, you can include an order by clause, using the column names (F0101.AN8) and ascending or descending order. Results are sorted by the requested columns, in the order they were added.	

Example - Data Service Java API

This example shows both a browse and a count of the F0101 table, including a query. The response from the browse is assembled into a class (not included) that was generated with the AIS Client Class Generator for F0101 data service. The count response is assembled into a simple HashMap and printed.

```
//add to the used capabilities
loginEnv.getUsedCapabilities().add(AISClientCapability.DATA_SERVICE);
loginEnv.getUsedCapabilities().add(AISClientCapability.DATA_SERVICE_ORDERBY);

//create a new DataReqeust
DataRequest f0101 = new DataRequest(loginEnv);

//Set table information, this is a browse of F0101
f0101.setDataServiceType(DataRequest.TYPE_BROWSE);
f0101.setTargetName("F0101");
f0101.setTargetType(DataRequest.TARGET_TABLE);
f0101.setFindOnEntry(FormRequest.TRUE);

//set return control ids, only these three columns will be in the response
```

```
f0101.setReturnControlIDs("F0101.AN8|F0101.ALPH|F0101.AT1");

//only return the first 10 records
f0101.setMaxPageSize("10");

//create a new query, for address numbers greater than 7000
Query greaterQ = new Query(loginEnv);
greaterQ.setAutoFind(true);
greaterQ.setMatchType(Query.MATCH_ALL);
greaterQ.addStringCondition("F0101.AN8", StringOperator.GREATER(), "7000");
f0101.setQuery(greaterQ);

//order byf0101.addOrderBy(loginEnv,"F0101", "AT1",
 OrderByDirection.ORDER_DIRECT_ASCENDING());f0101.addOrderBy(loginEnv,"F0101", "AN8",
 OrderByDirection.ORDER_DIRECT_DESCENDING())

//execute the data request
String response = JDERestServiceProvider.jdeRestServiceCall(loginEnv, f0101,
 JDERestServiceProvider.POST_METHOD, JDERestServiceProvider.DATA_SERVICE_URI);

//marshal the response to a formparent class generated by the class generator
DATABROWSE_F0101_FormParent f010Data
 =loginEnv.getObjectMapper().readValue(response,DATABROWSE_F0101_FormParent.class);

//modify the type to count, and get a count response for the same query
f0101.setDataServiceType(DataRequest.TYPE_COUNT);
String countresponse = JDERestServiceProvider.jdeRestServiceCall(loginEnv, f0101,
 JDERestServiceProvider.POST_METHOD, JDERestServiceProvider.DATA_SERVICE_URI);


//loop through the records in the response printing out the values
ArrayList<DATABROWSE_F0101_GridRow> rowSet =
 f010Data.getFs_DATABROWSE_F0101().getData().getGridData().getRowset();
if (rowSet.size() > 0)
{
    for (DATABROWSE_F0101_GridRow row: rowSet)
    {
System.out.println("Name: " + row.getSAlphaName_54().getValue());
System.out.println("Number: " + row.getMnAddressNumber_51().getValue());
System.out.println("Search Type: " + row.getSSchTyp_59().getValue());

System.out.println(" ");
    }
}
else
{
    fail("No Records in Reponse");
}

//marshal and print out the count response
HashMap countRespMap = loginEnv.getObjectMapper().readValue(countresponse, HashMap.class);

HashMap countMap = (HashMap)countRespMap.get("ds_F0101");
System.out.println(countMap.get("count"));
```

Determining the Maximum Records Returned in a Data Service

In a service request that returns rows in a grid, the AIS Server will return a maximum of 100 rows by default. If you want to return all records, include the following method in the form service request:

```
dataRequest.setMaxPageSizeUnlimited();
```

Data Service Data Aggregation (API 1.3.1 and EnterpriseOne Tools 9.2.0.2)

Data service data aggregation provides the capability to request an aggregation of values in a data service request. To perform aggregation functions over records from tables or business views, you must add both `dataservice` and `dataServiceAggregation` to the used capabilities list before using the aggregation APIs.

The following aggregation information is sent in the DataRequest object attribute:

```
aggregation, AggregationInfo object
```

You can also include a query in a data service data aggregation. See *Query with Aggregation* in this guide for more information.

Aggregation Arrays

An aggregation consists of the following three arrays of objects:

- *aggregations Array*
- *groupBy Array*
- *orderBy array*

aggregations Array

An array of columns with their associated aggregation type.

groupBy Array

An array of columns to group by.

Starting with API 1.4.2 and EnterpriseOne Tools 9.2.1, if grouping by a date field, you have the following additional options:

- Use the specialHandling field of the column in the groupBy array to indicate desired date formatting for the date groups. Possible specialHandling values are:
  - "User" – Uses the EnterpriseOne user's preferred date format.
  - "CALQTR" – Uses the four digit year and two digit month format, for example 2016-10.
  - SimpleDateFormat - Uses the simple date format that you supply, such as yyyy-MM-dd. Refer to the following Java documentation for the types of date format strings:

    https://docs.oracle.com/javase/7/docs/api/java/text/SimpleDateFormat.html
- Use APIs to create group by requests with these values:

```
//add a group by for users date preferenceaggregation.addAggregationGroupBy("F4211",
"DRQJ",AggregationInfo.DATE_USER_FORMATED)

//OR add a group by with SimpleDateFormatSimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
aggregation.addAggregationGroupBy("F4211", "DRQJ",sdf);
```

By default, the output of date groups is milliseconds.

The following code is an example of JSON for an aggregation object with a special date format used for groupBy:

```
"aggregation" : {
    "aggregations" : [ {
      "column" : "UPRC",
      "aggregation" : "SUM"
    }, {
      "column" : "*",
      "aggregation" : "COUNT"
    } ],
    "groupBy" : [ {
      "column" : "F4211.DRQJ",
      "specialHandling" : "yyyy-MM-dd"
    } ],
    "orderBy" : [ {
      "column" : "F4111.DRQJ",
      "direction" : "ASC"
    } ]
  }
```

orderBy array

An array of columns to order by with the direction. There are two possible directions to order by:

- Ascending, which uses the following constant:

  ```
  OrderByDirection.ORDER_DIRECT_ASCENDING()
  ```

- Descending, which uses the following constant:

  ```
  OrderByDirection.ORDER_DIRECT_DESCENDING()
  ```

You can also order by an aggregation result.

Aggregation Types

The following table describes the seven column-specific aggregation types that are available. You can combine more than one type in a single request. Based on the results of a find or query, multiple aggregations can be performed over multiple columns.

Column-specific Aggregation Types

Aggregation	Constant
Sum	`AggregationType.AGG_TYPE_SUM()`
Minimum	`AggregationType.AGG_TYPE_MIN()`

Aggregation	Constant
Maximum	`AggregationType.AGG_TYPE_MAX()`
Average	`AggregationType.AGG_TYPE_AVG()`
Count Distinct	`AggregationType.AGG_TYPE_COUNT_DISTINCT()`
Average Distinct	`AggregationType.AGG_TYPE_AVG_DISTINCT()`
Sum Distinct	`AggregationType.AGG_TYPE_SUM_DISTINCT()`

In addition, there is an aggregation type for performing a count called Count* that is available through the following API:

```
AggregationInfo_addCount();
```

Example of Coding an Aggregation Type Data Request

Example - Calling an Aggregation Type Data Request shows how to call an aggregation type data request. In the example, two data requests are sent in a single batch to the AIS Server. The first one is an aggregation of columns in V0101 without a Groupby array. The second is an aggregation request over F060116 with a Groupby array.

At the end of the data request, the AggregationResponseHelper methods are used to get the specific aggregations from the responses. If there is a Groupby array in the response, the helper will return an ArrayNode, which you will have to iterate with to get individual aggregate values for each group. Specific AggregationResponseHelper methods are provided for consuming a batch response.

Example - Calling an Aggregation Type Data Request

```
//Set used capabilities in login environment
 loginEnv.getUsedCapabilities().add(AISClientCapability.DATA_SERVICE);
 loginEnv.getUsedCapabilities().add(AISClientCapability.DATA_SERVICE_AGGREGATION);

 //create a new data request for V0101
 DataRequest dataAggregation = new DataRequest(loginEnv);

 //set type to aggregation
 dataAggregation.setDataServiceType(DataRequest.TYPE_AGGREGATION);
 dataAggregation.setTargetName("V0101");
 dataAggregation.setTargetType(DataRequest.TARGET_VIEW);

 //create aggergation info object
 AggregationInfo aggregation = new AggregationInfo(loginEnv);

 //add desired aggregations
 aggregation.addAggrigationColumn("AN8", AggregationType.AGG_TYPE_SUM());
 aggregation.addAggrigationColumn("AN8", AggregationType.AGG_TYPE_AVG());
 aggregation.addAggrigationColumn("AT1", AggregationType.AGG_TYPE_COUNT_DISTINCT());
 aggregation.addCount(); //this is for COUNT *


 //add aggregation to request
 dataAggregation.setAggregation(aggregation);
```

```
//query can be combined with aggregation
Query an8Query = new Query(loginEnv);
an8Query.setAutoFind(true);
an8Query.setMatchType(Query.MATCH_ALL);

an8Query.addNumberCondition("F0101.AN8", NumericOperator.LESS(), 6001);
dataAggregation.setQuery(an8Query);


//create a second datat request for F060116 aggregatoin
DataRequest dataAggregation2 = new DataRequest(loginEnv);
dataAggregation2.setFindOnEntry(true);
dataAggregation2.setDataServiceType(DataRequest.TYPE_AGGREGATION);
dataAggregation2.setTargetName("F060116");
dataAggregation2.setTargetType(DataRequest.TARGET_TABLE);

//create aggregation info object and add desired aggredgations
AggregationInfo aggregation2 = new AggregationInfo(loginEnv);
aggregation2.addAggrigationColumn("SAL", AggregationType.AGG_TYPE_SUM());
aggregation2.addAggrigationColumn("SAL", AggregationType.AGG_TYPE_AVG());
aggregation2.addAggrigationColumn("SAL", AggregationType.AGG_TYPE_AVG_DISTINCT());
aggregation2.addAggrigationColumn("SAL", AggregationType.AGG_TYPE_MAX());
aggregation2.addAggrigationColumn("SAL", AggregationType.AGG_TYPE_MIN());
aggregation2.addAggrigationColumn("SAL", AggregationType.AGG_TYPE_SUM_DISTINCT());
aggregation2.addCount();
aggregation2.addAggrigationColumn("AN8", AggregationType.AGG_TYPE_COUNT_DISTINCT());

//add desired group by columnst
aggregation2.addAggrigationGroupBy("HMCO");
aggregation2.addAggrigationGroupBy("HMCU");

//add desired order by with direction
aggregation2.addAggrigationOrderBy("HMCO", OrderByDirection.ORDER_DIRECT_DESCENDING());

//set the aggregation info in the request
dataAggregation2.setAggregation(aggregation2);


//add these two requests to a batch data request
BatchDataRequest batchDataRequest = new BatchDataRequest(loginEnv);
batchDataRequest.getDataRequests().add(dataAggregation);
batchDataRequest.getDataRequests().add(dataAggregation2);


//call the service
String response = JDERestServiceProvider.jdeRestServiceCall(loginEnv, batchDataRequest,
JDERestServiceProvider.POST_METHOD, JDERestServiceProvider.DATA_SERVICE_URI);


//get the number of distinct AT1 values
JsonNode distinct = AggregationResponseHelper.getSimpleAggregateValueBatch(response,
"V0101", 0, AggregationType.AGG_TYPE_COUNT_DISTINCT(), "AT1");
if (distinct != null)
{
    System.out.println("AT1 Distinct: " + distinct.asInt());
}

//get the count *
System.out.println("Count: " +AggregationResponseHelper.getCountBatch(response, "V0101",
0));
```

```
//get the grouped aggregations from the F060116 response, loop trhough and add the
 specific one (average to a hash map by group by)
 ArrayNode array = AggregationResponseHelper.getAggregateValuesArrayBtach(response,
 "F060116", 1);
 HashMap<String, Object> chartMap = new HashMap<String, Object>();
 for (Iterator groups = array.iterator(); groups.hasNext();)
 {
     JsonNode aGroup = (JsonNode) groups.next();
     JsonNode groupByInfo = aGroup.get(AggregationResponseHelper.GROUP_BY);
     JsonNode average = aGroup.get("SAL_SUM");

     chartMap.put(groupByInfo.get("HMCO").asText().trim() + "-" +
 groupByInfo.get("HMCU").asText().trim(), average.asDouble());
 }

 //print the map
 System.out.println("SAL SUM by HMCO/HMCU Map: " + chartMap);
```

Data Service Data Aggregation - Currency Decimals Support (API 1.3.3 and EnterpriseOne Tools 9.2.0.3)

The **`AggregationInfo.CurrencyProcessing`** class provides the capability to account for currency when performing data aggregation. This class enables you to specify the proper currency trigger for the data, which is originally defined in the table event rules in EnterpriseOne.

Currency data aggregation applies the correct number of decimal places to aggregated data based on the currency of the data. Use this class only if the table you are aggregating over has a currency table trigger in EnterpriseOne, and the columns you are aggregating are the currency columns processed in the table trigger. If you do not define currency processing in your request, numeric aggregated fields are given the number of display decimals defined in the data dictionary for that field. For more information about how currency is configured in EnterpriseOne, see "Using Currency" in the *JD Edwards EnterpriseOne Tools Development Guidelines for Application Design Guide* .

The following information is required for decimal processing in currency data aggregation:

- **Processing type**. You can choose from the following seven different processing modes, which are defined as constants in the AggregationInfo.CurrencyProcessing class:
    - **COMPANY**. Applies the currency code that is defined for the company in the key values.
    - **CURRENCYCODE**. Applies the currency code defined in the key values.
    - **MCU**. Applies the currency code of the business unit defined in the key values.
    - **AID**. Applies the currency code of the account ID defined in the key values.
    - **LEDGERTYPE_COMPANY_CURRENCYCODE**. Applies the currency of either the ledger type, company, or currency code (in this order) defined in the key values.
    - **LEDGERTYPE_CURRENCYCODE**. Applies the currency of either the ledger type or currency code (in this order) defined in the key values.
    - **STATIC**. Applies a single currency code set as a string in the currency code field.
- **Key columns**. Depending on the specified processing type, add key field columns from the table in the expected order.
- **Currency columns**. These are the columns that decimal processing will be applied to. You only need to add a column here once, even if you have several aggregations over that column. Identify each distinct column

to which the currency processing should apply. If you do not identify currency columns here, then currency processing will be applied to all columns identified as currency columns in the data dictionary. A currency column is defined in the data dictionary with class CURRENCY.

- **Currency code**. This field is used only for STATIC currency processing. Use it to set a string value (for example "USD") for the currency code to be used for all of the currency columns.
- **As If Currency**. (API 1.4.2 and EnterpriseOne Tools 9.2.1) This field is used to indicate a single currency for all of the response values. The exchange rate table value for the date specified (or today's date if not specified) will be used to calculate the values for the specified currency. In the response, results will be grouped only by defined "group by" fields because only one currency is present.
- **As If Currency Date**. (API 1.4.2 and EnterpriseOne Tools 9.2.1) Specify a date to determine which exchange rate is used for currency conversions to the 'as if' currency. If not specified, the current date is used.

When the proper currency processing is defined, it effects the result set received. At minimum, the data will always be grouped by the key columns defined in the currency processing. This is in addition to any other "group by" columns you have requested, if any.

Currency Processing Warning

If currency processing cannot determine the currency decimals to apply to a currency column, the output will display a message that currency was not processed. In this case, display decimals defined in the data dictionary for the column will be applied to the value instead of currency decimals. The following example shows the message that appears when currency is not processed:

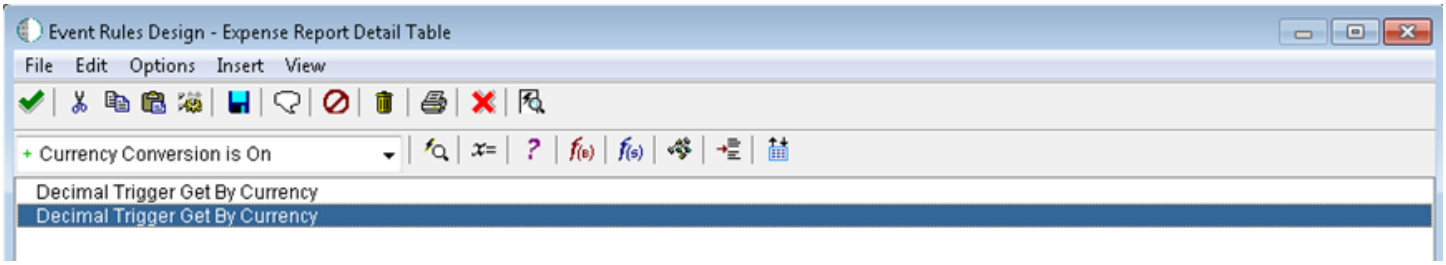
```
"groupBy" : {
      "F0911Z1.LT" : "AA",
      "F0911Z1.CRCD" : "    "
    },
    "F0911Z1.AA_SUM" : 3569.07,
    "F0911Z1.AA_SUM_CURRENCYINFO" : {
      "processedCurrency" : false
    },
    "COUNT" : 47

  }
```

Example of Coding for Currency

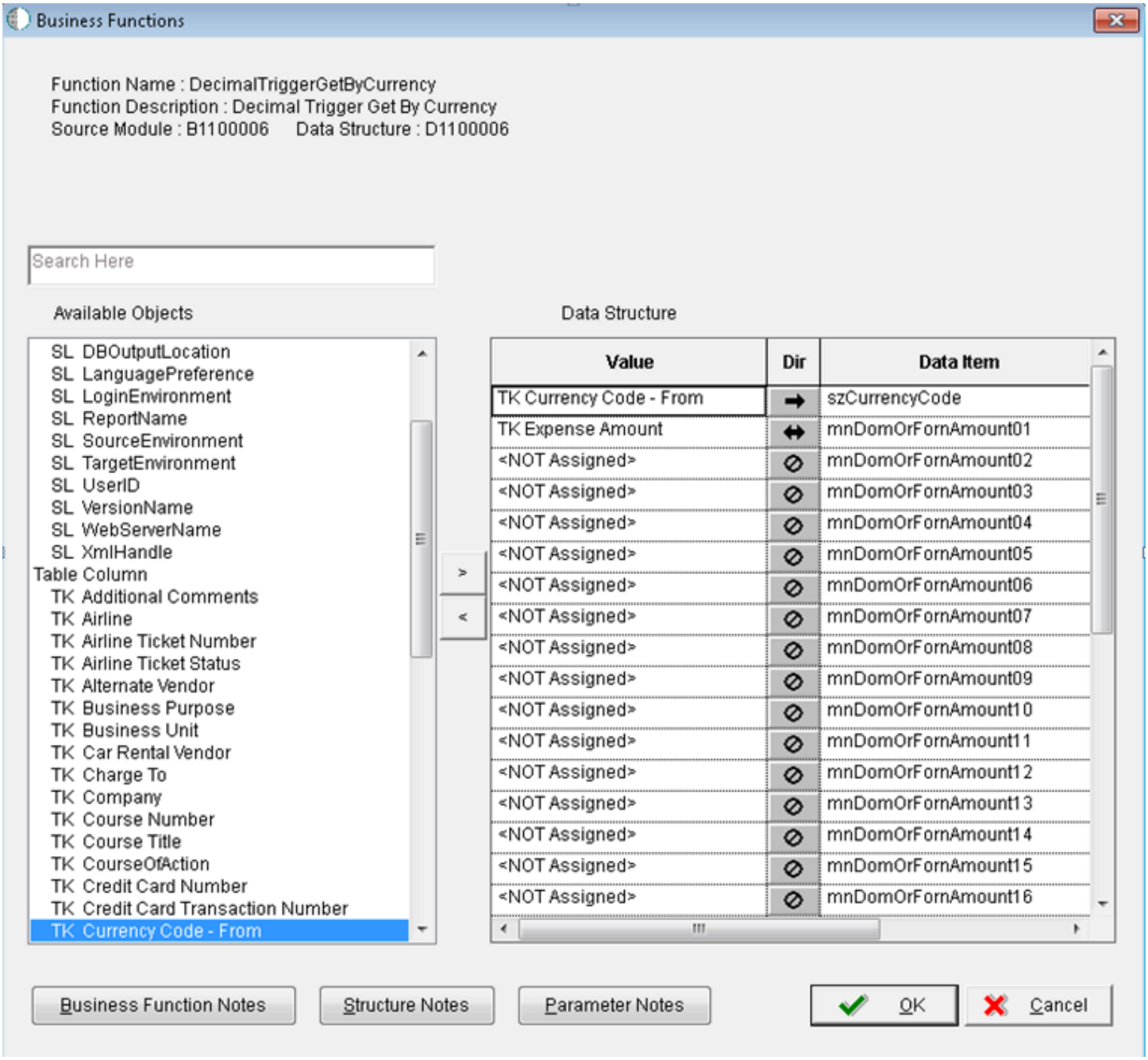
This section provides an example of how to identify the currency processing type for a column in EnterpriseOne, and then it describes how to specify that currency processing type in the data service so that the proper currency is applied to the data aggregation.

The aggregation in this example uses the Expense Report Detail Table (F20112). The F20112 table has two business functions defined in the table event rules, one that processes domestic amounts and another that processes foreign amounts. The amount used in this example aggregation is a foreign amount, specifically the EXPFAMT (Expense Amount).

The following image shows Event Rules Design in EnterpriseOne with the second business function selected because this is the one that operates on EXPFAMT.



In this function, the Expense Amount (F20112.EXPFAMT) field is processed based on a single key value, a currency code Currency Code - From (F20112.CRCD), as shown in the following image:



The following information has been identified in EnterpriseOne, which is enough information to code the data aggregation request:

- Type = `CURRENCYCODE`
- Key column = `CRCD`
- Currency column = `EXPFAMT`

The following example code shows how to apply the preceding values to the data service request so that the request applies currency to the data aggregation:

```
//specify all three used capabilities needed for the request
loginEnv.getUsedCapabilities().add(AISClientCapability.DATA_SERVICE);
loginEnv.getUsedCapabilities().add(AISClientCapability.DATA_SERVICE_AGGREGATION);
loginEnv.getUsedCapabilities().add(AISClientCapability.AGGREGATION_CURRENCY_DECIMAL);

//create the request for the F20112 table, aggregation type
DataRequest dataAggregation = new DataRequest(loginEnv);
dataAggregation.setDataServiceType(DataRequest.TYPE_AGGREGATION);
dataAggregation.setFindOnEntry(true);
dataAggregation.setTargetName("F20112");
dataAggregation.setTargetType(DataRequest.TARGET_TABLE);
//define the aggregation, we are requesting a sum and average of the EXPFAMT
AggregationInfo aggregation = new AggregationInfo(loginEnv);
aggregation.addAggregationColumn("F20112.EXPFAMT", AggregationType.AGG_TYPE_SUM());
aggregation.addAggregationColumn("F20112.EXPFAMT", AggregationType.AGG_TYPE_AVG());
dataAggregation.setAggregation(aggregation);

//group by employee
aggregation.addAggregationGroupBy("F20112. EMPLOYID");

//create a currency processing object
AggregationInfo.CurrencyProcessing currency = new
 AggregationInfo.CurrencyProcessing(loginEnv);
//select the CURRENCYCODE type of processing
currency.setType(AggregationInfo.CurrencyProcessing.CURRENCYCODE);
//add the key column of CRCD
currency.addToKeyCols("F20112.CRCD");
//add the currency column to be processed as EXPFAMT
currency.addToCurrencyCols("F20112.EXPFAMT");
aggregation.setCurrency(currency);

String response =
    JDERestServiceProvider.jdeRestServiceCall(loginEnv, dataAggregation,
 JDERestServiceProvider.POST_METHOD,
      JDERestServiceProvider.DATA_SERVICE_URI);

//process the response, which will be by employee and currency code because the request
 included group by of employee and the currency processing key was currency code

ArrayNode array = AggregationResponseHelper.getAggregateValuesArray(response, "F20112");

System.out.printf("%-10s %10s %10s %n", "EE ID", "Average","Currency");
System.out.printf("%-10s %10s %10s %n", "----", "------","-------");
for (Iterator groups = array.iterator(); groups.hasNext();) {
JsonNode aGroup = (JsonNode) groups.next();
JsonNode groupByInfo = aGroup.get(AggregationResponseHelper.GROUP_BY);
JsonNode average = aGroup.get("F20112.EXPFAMT_AVG");
JsonNode curencyCode = aGroup.get("currencyCode");
System.out.printf("%-10s %10s %10s %n",
 groupByInfo.get("F20112.EMPLOYID").asText().trim(),
 average.asText(),curencyCode.asText());
        }
```

The following image shows the charting of data used in this example:

EE ID	Average	Currency
6002	235.53	USD
6001	64387	JPY
6001	343.02	USD
5651	71.35	USD
5127	1422.5	USD
7702	175.4	CAD

To better understand the processing of the response, the JSON response looks like this:


```
{
  "ds_F20112" : {
    "output" : [ {
      "groupBy" : {
        "F20112.EMPLOYID" : 6002
      },
      "F20112.EXPFAMT_SUM" : 3532.88,
      "F20112.EXPFAMT_AVG" : 235.53,
      "currencyCode" : "USD"
    }, {
      "groupBy" : {
        "F20112.EMPLOYID" : 6001
      },
      "F20112.EXPFAMT_SUM" : 128773,
      "F20112.EXPFAMT_AVG" : 64387,
      "currencyCode" : "JPY"
    }, {
      "groupBy" : {
        "F20112.EMPLOYID" : 6001
      },
      "F20112.EXPFAMT_SUM" : 343.02,
      "F20112.EXPFAMT_AVG" : 343.02,
      "currencyCode" : "USD"
    }, {
      "groupBy" : {
        "F20112.EMPLOYID" : 5651
      },
      "F20112.EXPFAMT_SUM" : 214.06,
      "F20112.EXPFAMT_AVG" : 71.35,
      "currencyCode" : "USD"
    }, {
      "groupBy" : {
        "F20112.EMPLOYID" : 5127
      },
      "F20112.EXPFAMT_SUM" : 2845.0,
      "F20112.EXPFAMT_AVG" : 1422.5,
      "currencyCode" : "USD"
    }, {
      "groupBy" : {
        "F20112.EMPLOYID" : 7702
      },
      "F20112.EXPFAMT_SUM" : 175.4,
      "F20112.EXPFAMT_AVG" : 175.4,
      "currencyCode" : "CAD"
```

```
     }
    ]
   }
}
```

As If Currency Coding Example (API 1.4.2 and EnterpriseOne Tools 9.2.1)

The following example code shows what the code in *Example of Coding for Currency* would look like using As If currency:

```
//create a currency processing object
AggregationInfo.CurrencyProcessing currency = new
 AggregationInfo.CurrencyProcessing(loginEnv);
//select the CURRENCYCODE type of processing
currency.setType(AggregationInfo.CurrencyProcessing.CURRENCYCODE);
//add the key column of CRCD
currency.addToKeyCols("F20112.CRCD");
//add the currency column to be processed as EXPFAMT
currency.addToCurrencyCols("F20112.EXPFAMT");
currency.setAsIfCurrency("USD");
aggregation.setCurrency(currency);
```

The line highlighted in bold is the Java code line for As If currency, which was added to the currency processing object. The code line specifies that all values should be changed to USD.

The following code shows the JSON output for this example:

```
{
    "ds_F20112": {
        "output": [
            {
                "groupBy": {
                    "F20112.EMPLOYID": 6002
                },
                "F20112.EXPFAMT_SUM": 5232.88,
                "F20112.EXPFAMT_AVG": 290.72,
                "currencyCode": "USD",
                "currencyDecimals": 2
            },
            {
                "groupBy": {
                    "F20112.EMPLOYID": 6001
                },
                "F20112.EXPFAMT_SUM": 1218.42,
                "F20112.EXPFAMT_AVG": 609.21,
                "currencyCode": "USD",
                "currencyDecimals": 2
            },
            {
                "groupBy": {
                    "F20112.EMPLOYID": 5651
                },
                "F20112.EXPFAMT_SUM": 214.06,
                "F20112.EXPFAMT_AVG": 71.35,
                "currencyCode": "USD",
                "currencyDecimals": 2
            },
            {
                "groupBy": {
                    "F20112.EMPLOYID": 5127
```

```
                },
                "F20112.EXPFAMT_SUM": 2845,
                "F20112.EXPFAMT_AVG": 1422.5,
                "currencyCode": "USD",
                "currencyDecimals": 2
            },
            {
                "groupBy": {
                    "F20112.EMPLOYID": 7702
                },
                "F20112.EXPFAMT_SUM": 918,
                "F20112.EXPFAMT_AVG": 306,
                "currencyCode": "USD",
                "currencyDecimals": 2
            }
        ]
    }
}
```


The following image shows the charting of the data in this example:

EE ID	Average	Currency
6002	290.72	USD
6001	609.21	USD
5651	71.35	USD
5127	1422.5	USD
7702	306.0	USD

Understanding the Preference Service (API 1.3.1 and EnterpriseOne Tools 9.2.0.2)

The preference service enables an AIS client to save and retrieve user-level information that is stored by user, role, or *PUBLIC in the User Overrides Table (F98950) in EnterpriseOne.

The keys to the information are the Type (UOTY), User ID (USER), Sequence (SEQ) and Object Name (OBNM). The data is a string and is stored in the blob column BINDTA. All preference records are written with Type=PS.

Starting with AIS Client Java API 1.4.2 and EnterpriseOne Tools 9.2.1, the following four additional key fields allow for more distinct records: Form Name (FMNM), DelimitedControlIDs (IDLST), Version (VERS), and Language (LNGP). See *Example - Preference Service Java API* for a code example with these fields.

The AIS Client Java API enables you to save a serialized HashMap of data to this table using the methods in the PreferencesService object. For the get operation, the data is de-serialized to a HashMap and returned.

It is important to note that the data is stored as serialized values. For example, a Date object will be stored as the Long time, and you must convert it back to a date after the response is returned.

If a client application has multiple sets of data that need to be saved separately, a sequence field is available as a key. If you do not specify a sequence, the default sequence is zero.

Set and put actions are performed on behalf of the logged in user, established with the LoginEnvironment. All records for the users are stored in the USER column. You can use the User Overrides application (P98950) in EnterpriseOne to

manage the records in the User Overrides Table (F98950). In P98950, you can copy records to different roles including *PUBLIC. The record will be retrieved from P98950 based on the user, role, or *PUBLIC hierarchy.

Example - Preference Service Java API

```
//Add the preference service to the used capabilities
loginEnv.getUsedCapabilities().add(AISClientCapability.PREFEERNCE_SERVICE);
//create a new preference service object used to call the service
PreferenceService prefService = new PreferenceService(loginEnv);

//create a hashmap to store the valuse
HashMap<String,Object> preferenceData = new HashMap<String,Object>();

preferenceData.put("pref1", "Preference 1 String");
preferenceData.put("pref2", new BigDecimal("15.45"));
preferenceData.put("pref3",8);
preferenceData.put("pref4", new Date());


//call setPreferences to write the preference for the current logged in user based on the
 client id (Object Name), no sequence used so it will be saved as zero sequence
prefService.setPreferences("AIS_CLIENT", preferenceData);

// create a hashmap to store the retrieved values
HashMap<String,Object> preferenceDataOut
=prefService.getPreferences("AIS_CLIENT ");

//print response showing values were received
out.println("Recieved: " + preferenceDataOut);
//get individual value
out.println("Recieved Pref 2: " + preferenceDataOut.get("pref2"));
```

Example - Preference Service with Additional Key Fields (API 1.4.2 and EnterpriseOne Tools 9.2.1)

```
        //Add the preference service to the used capabilities
        loginEnv.getUsedCapabilities().add(AISClientCapability.PREFEERNCE_SERVICE);
        //create a new preference service object used to call the service
        PreferenceService prefService = new PreferenceService(loginEnv);

        //create a hashmap to store the valuse
        HashMap<String,Object> preferenceData = new HashMap<String,Object>();

        preferenceData.put("pref1", "Preference 1 String");
        preferenceData.put("pref2", new BigDecimal("15.45"));
        preferenceData.put("pref3",8);
        preferenceData.put("pref4", new Date());


        //call setPreferences to write the preference for the current logged in user based
 on the client id (Object Name), no sequence used so it will be saved as zero sequence
 additional fields of Form Name, ID List, Version and Language are passed.  You can pass
 null for any of these values also, and they will not be populated.
        prefService.setPreferences("AIS_CLIENT","W01012A", "10","ZJDE0001","E",
 preferenceData);
```

```
        // create a hashmap to store the retrieved values
        HashMap<String,Object> preferenceDataOut=
        prefService.getPreferences("AIS_CLIENT","W01012A", "10","ZJDE0001","E");

        //print response showing values were received
        out.println("Recieved: " + preferenceDataOut);
        //get individual value
        out.println("Recieved Pref 2: " + preferenceDataOut.get("pref2"));
```

Watchlist Service (API 1.4.0 and EnterpriseOne Tools 9.2.0.3)

Use the following watchlist URI to invoke a watchlist:

`/jderest/watchlist`

In the WatchlistRequest, you provide a watchlist ID or watchlist object name, which you can locate in the information about the watchlist in the EnterpriseOne web client.

Additionally, in a watchlist request, you can force an update so that the watchlist data is fetched from the database instead of the cache. You can request that the watchlist be set to dirty, so the next request will be fetched from the database and not from cache.

Use the WatchListResponse object to marshal the JSON response to an object. This enables access to all of the information about the watchlist including the record count and all threshold information.

Example - Watchlist Request

```
WatchListRequest wlr = new WatchListRequest(loginEnv);
wlr.setWatchlistObjectName("OVW4210E_1512070001JDE");


String response =
            JDERestServiceProvider.jdeRestServiceCall(loginEnv, wlr,
 JDERestServiceProvider.POST_METHOD,
 JDERestServiceProvider.WATCHLIST_SERVICE);

        if (!response.contains("invalid")) {
            WatchListResponse wlrs = loginEnv.getObjectMapper().readValue(response,
 WatchListResponse.class);
            System.out.println("Watchlist Form: " + wlrs.getFormtitle());
            System.out.println("Watchlist Description: " + wlrs.getDescription());
            System.out.println("Watchlist Count: " + wlrs.getRowcount().getRecords());
            System.out.println("Watchlist Warning: " + wlrs.isIsWarning());
            System.out.println("Watchlist Critical: " + wlrs.isIsCritical());
        }
```

Additional Supported Output Types for Form Service and Data Service (API 1.3.1 and EnterpriseOne Tools 9.2.0.2)

You can use the outputType parameter to enable the requester to control the format of the JSON response from either a form service or a data service.

Valid values in the outputType parameter for the supported output types are:

- GRID_DATA_OUTPUT_TYPE

  This output type returns only data in the grid in simplified name-value pairs.

- VERSION2_OUTPUT_TYPE

  This output type returns column-level information moved out of the grid row cell to a columnInfo section. Only cell specific information in each grid row cell is returned. Only non-default values are returned for fields on form and in the grid.

Note: For Application Stack calls, the output type must be set at the top level ApplicationStack object; all responses for that stack object will have the indicated output type. For example:

```
ApplicationStack appStackObj = new ApplicationStack();
appStackObj.setOutputType(FormRequest.VERSION2_OUTPUT_TYPE);
```

Grid Data Output Type (API 1.3.1 and EnterpriseOne Tools 9.2.0.2)

Grid Data output type returns data in simple name value pairs, with no additional metadata describing each field.

Form Request Usage

```
FormRequest formRequest = new FormRequest(loginEnv);
formRequest.setOutputType(FormRequest.GRID_DATA_OUTPUT_TYPE);
```

Example - Form Request Response

```
{
    "fs_P0801_W0801A": {
        "title": "Work With Employee Information",
        "data": {
            "gridData": {
                "columns": {
                    "z_AN8_14": "Employee No",
                    "z_ALPH_15": "Alpha Name",
                    "z_HMCO_24": "Home Company",
                    "z_DST_409": "Date Started"
                },
                "rowset": [
                    {
                        "z_DST_409": "20110410",
                        "z_AN8_14": 6002,
                        "z_HMCO_24": "00001",
                        "z_ALPH_15": "Abbott, Dominique"
```

```
                    },
                    {
                        "z_DST_409": "20100418",
                        "z_AN8_14": 6044,
                        "z_HMCO_24": "00001",
                        "z_ALPH_15": "Abrams, Brooke"
                    },
                    {
                        "z_DST_409": "20170302",
                        "z_AN8_14": 6078,
                        "z_HMCO_24": "00001",
                        "z_ALPH_15": "Aiken, Gwen"
                    },
                    {
                        "z_DST_409": "20130101",
                        "z_AN8_14": 8985155,
                        "z_HMCO_24": "00200",
                        "z_ALPH_15": "Allan, Murray"
                    },
                    {
                        "z_DST_409": "19720613",
                        "z_AN8_14": 7747,
                        "z_HMCO_24": "00077",
                        "z_ALPH_15": "Almeida, Wendy"
                    }
                ],
                "summary": {
                    "records": 5,
                    "moreRecords": true
                }
            }
        },
        "errors": [],
        "warnings": []
    },
    "stackId": 1,
    "stateId": 1,
    "rid": "e62ee1e430768855",
    "currentApp": "P0801_W0801A_ZJDE0001",
    "timeStamp": "2015-09-03:10.44.47",
    "sysErrors": []
}
```


Data Request Usage

```
DataRequest f0101 = new DataRequest(loginEnv);
f0101.setOutputType(DataRequest.GRID_DATA_OUTPUT_TYPE);
```

Example - Data Request Response


```
{
    "fs_DATABROWSE_F0101": {
        "title": "Data Browser - F0101 [Address Book Master]",
        "data": {
            "gridData": {
                "columns": {
                    "F0101_AN8": "Address Number",
                    "F0101_ALPH": "Alpha Name",
                    "F0101_AT1": "Sch Typ"
                },
```

```
                "rowset": [
                    {
                        "F0101_AT1": "O",
                        "F0101_AN8": 1,
                        "F0101_ALPH": "Financial/Distribution Company"
                    },
                    {
                        "F0101_AT1": "O",
                        "F0101_AN8": 9,
                        "F0101_ALPH": "Multi-Site Target Company"
                    },
                    {
                        "F0101_AT1": "O",
                        "F0101_AN8": 20,
                        "F0101_ALPH": "Marketing Company"
                    },
                    {
                        "F0101_AT1": "F",
                        "F0101_AN8": 27,
                        "F0101_ALPH": "Eastern Area Distribution Center"
                    },
                    {
                        "F0101_AT1": "O",
                        "F0101_AN8": 28,
                        "F0101_ALPH": "Prueba - Argentina - 28"
                    }
                ],
                "summary": {
                    "records": 5,
                    "moreRecords": true
                }
            }
        },
        "errors": [],
        "warnings": []
    },
    "stackId": 2,
    "stateId": 1,
    "rid": "e62ee1e430768855",
    "currentApp": "DATABROWSE_F0101",
    "timeStamp": "2015-09-03:10.44.48",
    "sysErrors": []
}
```


Example - Using Jackson Libraries to Iterate Through Rows and Get Values

```
JsonNode node  = loginEnv.getObjectMapper().readTree(response);
JsonNode array =
 node.path("fs_DATABROWSE_F0101").path("data").path("gridData").path("rowset");
for (Iterator<JsonNode> rows = array.iterator(); rows.hasNext();)
{
   JsonNode aRow = rows.next();
   System.out.println("Name: " +aRow.get("F0101_ALPH"));
}
```

Version2 Output Type (API 1.3.1 and EnterpriseOne Tools 9.2.0.2)

The AIS Client Class Generator version 2.0.0 supports generating classes for both the original output and the Version2 output type.

Form Request Usage

```
loginEnv.getUsedCapabilities().add(AISClientCapability.OUTPUT_TYPE);
formRequest.setOutputType(FormRequest.VERSION2_OUTPUT_TYPE);
```

Data Request Usage

```
loginEnv.getUsedCapabilities().add(AISClientCapability.OUTPUT_TYPE);
dataRequest.setOutputType(DataRequest.VERSION2_OUTPUT_TYPE);
```

When using the AIS Client Class Generator, make sure to choose Version 2 for the Output Version. You can use the classes generated with Version 2 in the same way as you used classes generated in the original version - Version 1.

Orchestration Support (API 1.1.0)

The AIS Server supports form service request calls from orchestrations. The AIS Server must be configured to work with orchestrations. See *"Prerequisites" in the JD Edwards EnterpriseOne Tools Orchestrator Guide for Studio Version 8 and Prior* for more information.

This section describes how to invoke the orchestration using the AIS Client API.

Orchestrator requests are stateless. The entire orchestration is executed in a single call and returns the results of the orchestration.

Example - Using JDE Standard Input Format for an Orchestration

This example uses the JDE Standard input format. The orchestration must be configured to accept this input format.

```
OrchestrationRequest req = new OrchestrationRequest(AIS_SERVER ,USER_NAME,
 PASSWORD,DEVICE_NAME);

req.setOrchestration("GetAddressBook_Simple");
req.getInputs().add(new OrchestrationInputValue("AddressBookNumber", "7500"));
req.getInputs().add(new OrchestrationInputValue("SearchType", "E"));


try{

  String output = req.executeOrchestrationRequest();

 //consume output, you can deserialize it to a class generated by the AIS Class Generator

}
catch(Exception e)
{
   //handle exceptions
```

```
}
```

Example - Using Generic Input for an Orchestration

This example uses the Generic input format. The orchestration must be configured to accept this input format.

```
OrchestrationRequest req = new OrchestrationRequest(AIS_SERVER ,USER_NAME,
 PASSWORD,DEVICE_NAME);

orchRequest.setOrchestration("AddCBM");

//populate values to send from this instance, simple name value pairs hash map
HashMap<String,String> vals = new HashMap<String,String>();
vals.setSerialNumber("02a0bd30-d883-11e4-b9d6-1681e6b88ec1");
Date jDate = new Date();
vals.put("date",String.valueOf(jDate.getTime()));
SimpleDateFormat sdf = new SimpleDateFormat("hh:mm:ss");
vals.put("time",sdf.format(jDate));
vals.put("temperature","201");
vals.put("description","Temp 201");

try
{

    String response = orchRequest.executeOrchestrationRequest(values);

//consume response, you can deserialize it to a class generated by the AIS Class Generator

}
catch(Exception e)
{
   //handle exceptions
}
```

Next Page Processing for Application Stack and Data Request (API 2.0.0 and EnterpriseOne Tools Release 9.2.1.2)

Use the "next link" capability to fetch data in easily manageable data sets over several successive service calls. This capability is available in version 2 AIS services. Therefore, you must include v2 in the URI to access the "next link" capability in an application stack or data request, for example:

http://<ais_server>:<port>/jderest/v2/appstack

http://<ais_server>:<port>/jderest/v2/dataservice

Next Page for Application Stack

An application stack response may include one or more grids. If any of the grids indicate more records are available ("moreRecords" : true), then the response will include a links section. *Example - Links Section in an Application Stack Response* shows a links section in an application stack response.

Example - Links Section in an Application Stack Response

```
"links" : [ {
    "rel" : "next",
    "href" : "http:// <ais_server>:<port>//jderest/v2/appstack/next?
stackId=1&stateId=1&rid=5b5e9d49dac52bb2&fullGridId=1&formOID=W0801A&token=044hq8nL%2FEHTb3dhe3XqPxl15jr
%2FoFlXX0yEOI%2FixZTNdU
%3DMDE5MDEzOTEyNTY5MTE3MzA1Njk3NjIxOTEwLjEzOS4xMTUuNTUxNDgyNTA5NTc2MjI5&outputType=GRID_DATA&returnControlIDs=1[14,15]",
    "context" : "1"
  } ]
```

To request the next page of data, use the URL provided in the links section to perform a POST or a GET operation for that URL.

To request the last page of data, use the URL provided in the links section to perform a POST or GET operation for that URL.

It is important to note that the size of each data set is determined up front in the first request to the grid in the form. The maxPageSize input parameter indicates the number of records to return with each call. For example, if you set this value to 10, you will receive 10 records in the first call and up to 10 more in every subsequent link call.

The period of time that the current data set remains open is determined by the resultSetTimeout setting for the EnterpriseOne HTML Server, which you can configure in Server Manager. The default for this setting is 60 seconds. Although unlikely to occur, you must consider the possibility that the result set will time out and throw an exception when calling the next link. In the case of a timeout, you will have to start over and re-run the original fetch to refresh the data set.

The following is an example of the JSON response for a timeout:

```
Status: 500
{
    "sysErrors": [
        {
            "TITLE": "JAS_MSG347: The query results expired.  Refresh them by clicking
 Find again.",
            "DESC": "JAS_MSG347: The query results expired.  Refresh them by clicking Find
 again."
        }
    ]
}
```

AIS Client API Next Page for Application Stack

In the AIS Client Java API 2.0.0, the Application Stack API contains objects for getting and executing a response link.

Example - Next Page Processing through the ApplicationStack API shows an application stack with a request for data from P0801 with maxPageSize set to 25. It will execute any next page links until the last record has been fetched and no links remain.

Example - Next Page Processing through the ApplicationStack API

```
    public void nextPageLinks() throws Exception
    {

        ApplicationStack appStackAddress = new ApplicationStack();
        FormRequest formRequest = new FormRequest(loginEnv);
        formRequest.setFormName("P0801_W0801A");
        formRequest.setVersion("ZJDE0001");
        formRequest.setReturnControlIDs("1[14,15]");
        formRequest.setFormServiceAction("R");
        formRequest.setMaxPageSize("25");
        formRequest.setFindOnEntry(true);


        //open P0801_W0801A
        String response = appStackAddress.open(loginEnv, formRequest);

  //Add Code here to marshal the response...

  //continue fetching more records until no more 'next' links are received
 while(appStackAddress.getLastAppStackResponse().getLinks()!=null &&
appStackAddress.getLastAppStackResponse().getLinks().isEmpty()
        && appStackAddress.getLastAppStackResponse().getLinks().get(0).getRel().equals("next"))
        {

            //get more
            response =
 appStackAddress.executeLink(loginEnv,appStackAddress.getLastAppStackResponse().getLinks().get(0));
  //Add Code here to marshal each response...

        }

        //close
        response = appStackAddress.close(loginEnv);

    }
```

In this example, if the result set times out, the system throws a JDERestServiceException with the following message:

JDE Rest Service Call Failed: Status: 500 {"sysErrors":[{"TITLE":"JAS_MSG347: The query results expired. Refresh them by clicking Find again.","DESC":"JAS_MSG347: The query results expired. Refresh them by clicking Find again."}]}

Next Page for Data Service

For data service calls, you must set the enableNextPageProcessing parameter to "true" in the first call if you expect to receive next page links. This keeps the data set open waiting to receive the next page requests. When configured in the first call, if there are additional records to be fetched, the Data Service response will include a links section, as shown in *Example - Links Section in a Data Service Response*.

Example - Links Section in a Data Service Response

```
"links": [   {
      "rel": "next",
      "href": "http://<ais_server>:<port>/jderest/v2/dataservice/next?
stackId=1&stateId=1&rid=48e401e5950224d&fullGridId=54&formOID=V0101&token=044hUE
%2B82ZUWZB7MLBBIwG6uX%2FgYKiZFEgsPV99bul%2FYjQ
%3DMDE5MDEyNTE3OTEzMDczODg5MzA4MDE3MTE0OC44Ny4xOS40NjE0ODI1MTE0NjM4NTQ
%3D&outputType=GRID_DATA&returnControlIDs=F0101.AN8|F0101.AT1|F0101.ALPH"
   }]
```

As with the application stack, the size of each data set is determined up front using the maxPageSize parameter in the first request to a grid in a form.

But data service next page processing has an additional request parameter called nextPageTimeInterval that effects a process in the background referred to as "data dripping." When this parameter is not used, data dripping occurs where data is fetched from the database silently in the background at five second intervals awaiting the next call. When the next call occurs, it fetches whatever is remaining up to the maxPageSize.

If you set nextPageTimeInterval to 1000 milliseconds or less, data dripping is turned off. You can still perform next calls, but the entire next section is fetched at the time of the next call.

If you set nextPageTimeInterval to a value greater than 1000, but less than the configured jdbj.ini resultSetTimeout, then the records will be silently fetched for the time interval provided.

In any case, it is possible that the result set will time out and the next link call will throw an exception. In the case of a timeout, you will have to start over and re-run the original fetch to refresh the data set.

The JSON response for a time out looks like this:

```
Status: 500
{
    "sysErrors": [
        {
            "TITLE": "JAS_MSG347: The query results expired.  Refresh them by clicking
 Find again.",
            "DESC": "JAS_MSG347: The query results expired.  Refresh them by clicking Find
 again."
        }
    ]
}
```

AIS Client API Next Page for Data Service

In the AIS Client Java API 2.0.0, the data service API contains objects for processing links in the response from a data service call.

shows a data service request with records fetched from F0101 in 100 record chunks and marshaled into an object. It will execute any next page links until the end of the record set is printed and no links remain.

Example

```
    public void nextPageDataRequest() throws Exception
    {
        DataRequest f0101 = new DataRequest(loginEnv);
        f0101.setDataServiceType(DataRequest.TYPE_BROWSE);
        f0101.setTargetName("F0101");
        f0101.setTargetType(DataRequest.TARGET_TABLE);
        f0101.setFindOnEntry(FormRequest.TRUE);
        f0101.setReturnControlIDs("F0101.AN8|F0101.ALPH|F0101.AT1");
        f0101.setMaxPageSize("100");
        f0101.setEnableNextPageProcessing(true);


        String response = JDERestServiceProvider.jdeRestServiceCall(loginEnv, f0101,
JDERestServiceProvider.POST_METHOD, JDERestServiceProvider.DATA_SERVICE_URI);
```

```
   //marshal to generated class
        DATABROWSE_F0101_FormParent f010Data =
 loginEnv.getObjectMapper().readValue(response, DATABROWSE_F0101_FormParent.class);

        //loop while there are next records, print out each 100 record set

        while(f010Data != null){
            ArrayList<DATABROWSE_F0101_GridRow> rowSet =
 f010Data.getFs_DATABROWSE_F0101().getData().getGridData().getRowset();
            if (rowSet.size() > 0)
            {
                for (DATABROWSE_F0101_GridRow row: rowSet)
                {
                    System.out.print("Number: " + row.getMnAddressNumber_51()+ ", ");
                    System.out.print("Name: " + row.getSAlphaName_52() + ", ");

                    System.out.println("Search Type: " + row.getSSchTyp_53());

                }
            }
   //stop the loop this might be the last one
            f010Data = null;

   //try to fetch the next set
            try
            {
                DataRequestLinks drLinks = loginEnv.getObjectMapper().readValue(response,
 DataRequestLinks.class);
                if (drLinks.getLinks() != null && drLinks.getLinks().size() > 0)
                {
                    response = drLinks.executeLink(loginEnv, drLinks.getLinks().get(0));
    //marshal each response
                    f010Data = loginEnv.getObjectMapper().readValue(response,
 DATABROWSE_F0101_FormParent.class);
                }

            }
            catch(JDERestServiceException e)
            {
                System.out.println(e.getMessage());
            }

        }
    }
```

In this example, it the result set times out, the system throws a JDERestServiceException with the following message:

```
JDE Rest Service Call Failed: Status: 500 {"sysErrors":[{"TITLE":"JAS_MSG347: The query results expired. Refresh them by clicking Find again.","DESC":"JAS_MSG347: The query results expired. Refresh them by clicking Find again."}]}
```

5 Glossary

AIS Server

A REST services server that when configured with an EnterpriseOne HTML Server, enables access to EnterpriseOne forms and data.

AIS Server capability

A behavior of the AIS Server that an AIS client can use to perform a particular EnterpriseOne task, such as update a grid record or fetch a processing option.

AIS client

An application that uses the AIS Server to communicate with EnterpriseOne.

AIS Server endpoint

An endpoint on the AIS Server that provides a service for the AIS client. An AIS client can access an AIS Server endpoint through a URL. In turn, the endpoint performs a particular service for the AIS client in EnterpriseOne.

AIS service

A service in an AIS Server endpoint. An AIS service interacts with EnterpriseOne based on input from an AIS client and provides a response in JSON format.

form service request

An AIS Server call that retrieves data from a form in EnterpriseOne. Form service requests, formatted as REST service calls that use POST, contain form service events or commands that invoke actions on an EnterpriseOne form.

instantiate

A Java term meaning "to create." When a class is instantiated, a new instance is created.

JDeveloper Project

An artifact that JDeveloper uses to categorize and compile source files.

JSON (JavaScript Object Notation)

A light-weight format used for the interchange of data between the AIS Server and EnterpriseOne.

processing option

A data structure that enables users to supply parameters that regulate the running of a batch program or report. For example, you can use processing options to specify default values for certain fields, to determine how information appears or is printed, to specify date ranges, to supply runtime values that regulate program execution, and so on.

QBE

An abbreviation for query by example. In JD Edwards EnterpriseOne, the QBE line is the top line on a detail area that is used for filtering data.

serialize

The process of converting an object or data into a format for storage or transmission across a network connection link with the ability to reconstruct the original data or objects when needed.