Costpoint Extensibility Designer Coding Guide

Costpoint Extensibility Designer Coding Guide

26 March 2026

Contents

Costpoint Extensibility Designer Coding Guide

Standards are necessary for ease of reference, maintenance and collaboration. This section provides standards to be followed for Java coding for Costpoint application and extensibility.

Java Standards

Costpoint development adopts the Java Naming Standards as the starting point for all coding conventions and standards.

Attention: For more information, visit the following web page:

http://www.oracle.com/technetwork/java/javase/documentation/codeconvtoc-136057.html

This section provides specific details that apply to Costpoint development. If the details are absent in this document, Java Naming Standards should be followed.

Package Structure

All package names are in lower case.

All Costpoint packages start with com.deltek.enterprise. There are four top branches underneath. They are **core**, **system**, **tools** and **application**. Packages for your extensibility will form the fifth branch named **extensions**.

Package	Description
com.deltek.enterprise.core	Private framework code. Extensibility developer may not use any of these packages.
com.deltek.enterprise.system	Private framework code except two packages in this branch that extensibility developer can use: applicationinterface and utils (see details later).
com.deltek.enterprise.tools	Private framework code. Extensibility developer may not use any of these packages.
com.deltek.enterprise.application	Private common or specific application code containing business rules. Extensibility developer may use this package.
com.deltek.enterprise.extensions	All code developed for application extensibility must reside under this package.

The diagram above shows that code in the extensions branch should only use packages from system.applicationinterface, and system.utils (not shown).

com.deltek.enterprise.extensions.<extensibility project id>

Under extensions package, the next level must be named the same as the extensibility project ID (created in the Extensibility Designer)

For example

com.deltek.enterprise.extensions.xt_project_y

We only enforce the structure up to this level.

Under the project level, we recommend you sub package this further into branches for ease of reference.

For example

- com.deltek.enterprise.extensions.xt_project_y.common (to be shared by all code under this project)
- com.deltek.enterprise.extensions.xt_project_y.unit1.common (to be shared by all code under unit1)
- com.deltek.enterprise.extensions.xt_project_y.unit1.symusr (specific code for extending the SYMUSR app under unit 1)

Class

The class name should always start with an upper case letter. Class names should be nouns, in mixed case with the first letter of each internal word capitalized. Keep your class names simple and descriptive. Do not use hyphen for class name.

For app specific java classes, developers should follow the below naming pattern:

<App ID> [Optional ChildRSName]<eventType>

Where eventType can be RsPopulate, ObjValidation, LineValidation, RSValidation, BeforeSave, or AfterSave. (see event plug-in section).

For example:

- com.deltek.enterprise.extensions.xt_project_y.unit1.symusr.SymusrRsPopulate.java
- com.deltek.enterprise.extensions.xt_project_y.unit1.symusr.SymusrObjValidation.java

App common helper file can be named as <App ID>common.

For example:

- com.deltek.enterprise.extensions.xt_project_y.unit1.symusr.SymusrCommon.java

Method

Methods should be verbs, in mixed case with the first letter lowercase and the first letter of each internal word capitalized.

For example

- validateHdrInfo
- computeExchgRates

Variable

Variable names should be short yet meaningful. The choice of a variable name should be mnemonic- that is, designed to indicate to the casual observer the intent of its use. One-character variable names should be avoided except for temporary "throwaway" variables.

Similar to method name, variable names are in mixed case with the first letter lowercase and the first letter of each internal word capitalized. Do not use underscores.

Variable holding value from a database column should assume the name of the database column (without the underscores).

For example:

- projId for column PROJ_ID
- vchrKey for column VCHR_KEY

Constant

The names of variables declared class constants and of ANSI constants should be all uppercase with words separated by underscores ("_").

For example:

- public static final short OPEN_APP
- public static final short CLOSE_APP

Comments

Comments for classes and for methods should follow the Javadoc guidelines.

Attention: For more information, please refer to the following site:

http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html

Indentation

All source code should be properly indented for ease of reading.

Java Logging

Avoid using System.out and use LoggerInterface to enable more flexibility and extensibility in logging information.

Attention: For more information, please refer to the LoggerInterface section.

Imports

When developing application java code, only two packages can be imported from the system branch:

- import com.deltek.enterprise.system.applicationinterface.*;
- import com.deltek.enterprise.system.utils.*;

You can import any packages from any common code that you **create for your extensions packages**.

- import com.deltek.enterprise.extensions.xt_project_y.common.*
- import java.util.*
- import java.lang.*
- import java.text.*

You can import these standard Java packages if you need them for development.

- import java.sql.SQLException;

Import this class if your class accesses the database (via the SqlManager class -- discussed later)

Note: If you are developing an Extension for Deltek's Cloud, those are the only packages/classes that you can import. If you would like to import other classes, please discuss it and get permission from Cloud Operations team.

If you are configuring Java IDE or compiling Java source files with the javac command, you would need to point it to or add it to your classpath main Costpoint classes folder in order to import other Costpoint classes. This folder is located under the main Costpoint folder: \deltek\costpoint\82\applications\enterprise\APP-INF\classes and most classes that are deployed with Costpoint are located in this folder. Your Extension classes should be placed in the same folder after they are compiled for Weblogic in order to load them at run-time. Please note that if you are making changes and recompiled your classes, you would need to restart the Weblogic nodes for the changes to become effective. That is also why it is required to restart the Weblogic nodes after you deploy extension(s) that have java classes in them.

Upon certain events triggered by the user, the framework will call the plug-in java classes registered by the developer in the Designer tool for that event.

For example, on Query, the framework will call the App Populate event plug-in. On tabbing out of a field (and user login with Field mode), the framework will call object validation event plug-in. On leaving a line to the next line (and user login with Field mode or Line mode), the framework will call line validation plug-in event (See plug-in event section for more details).

When calling the plug-in method, the framework will pass the handle to current screen (result set) the ResultSetInterface. From this handle, application code can access and manipulate data contained in that result set and perform calculation or validation as necessary. Data on the screen can be manipulated via the RowsetInterface while data in the database can be directly updated via the SqlManager interface.

Note: There are exceptions when the framework passes AppInterface and ActionInterface (discussed later).

system.applicationinterface.DEException

This class is a wrapper class for the root class java.lang.Exception. All application method must throw this exception instead of various types of java exception. The system will catch this exception and handle it gracefully without bringing down the entire session or even the application server.

system.applicationinterface.ResultSetInterface

ResultSetInterface is the most common interface between the system and the application java classes.

This interface is passed to the result set each time a plug-in method is called from the system.

Note: In some instances, appInterface or actionInterface is passed instead -- explained later.

This interface provides methods to get to objects on the current result set, the parent's or children result set.

Here are some commonly used methods provided by the interface (see javadoc for more explanations)

- getRowSet(): Returns the RowSetInterface to the row in context. You always need to get RowSetInterface before you can get to the data on the row.

- getApplication(): Returns the AppInterface to the app user is in.
- getSqlManager(Object o): Returns an instance of SqlManager which is used to access the database via SQL.
- getParentRS(): Returns the ResultSetInterface of the parent RS when the application contains more than one result set. Always check for null before using the returned ResultSetInterface (if user does not open the subtask screen, ResultSetInterface of the subtask may be null).
- getChild(String rsId): Returns the ResultSetInterface of the specified child RS. Always check for null before using the returned ResultSetInterface.
- findInit(int flagsOn,int flagsOff,boolean setContext): Returns an iterator so you can loop through rows in the result set.
- findFirstRow(int flagsOn,int flagsOff): Returns the number of the first row (each row has a number) found in the search.
- addObjectMessage(String objectId,String msgId, short msgType): Add a message and its severity. Set focus to the indicated object in the UI.
  Overload method:
  addObjectMessage(String objectId,String msgId, short msgType, String parms[]): Add a tokenized message and its severity and associate with an object.
- addLineMessage(String msgId, short msgType): Add a message and its severity. Set focus to the entire line in the UI.

Example: Validate a project ID and issue an error message for invalid project id

```
package com.deltek.enterprise.extensions.xt_project_y.mypjmbasic;
import com.deltek.enterprise.system.applicationinterface.DEException;
import com.deltek.enterprise.system.applicationinterface.CPConstants;
import com.deltek.enterprise.system.applicationinterface.ResultSetInterface;
import com.deltek.enterprise.system.applicationinterface.RowSetInterface;
public class MyPjmbasicObjectValidation {
public short validateProjId (ResultSetInterface rsI) throws DEException {

/* Initialize return code to OK */
short retCode = CPConstants.CP_OK;
/* Get to RowSetInterface from ResultSetInterface */
RowSetInterface roI = rsI.getRowSet();
/* Get data on the row */
String projID = roI.getStringValue("PROJ_ID");
/* Call some method to validate the value of project ID */
if (!checkProjID(projID)) {


/* Return an error if project ID is not validated */
rsI.addObjectMessage("PROJ_ID","CP_PROJ_INVALID", rsI.ERROR);
retCode = rsI.ERROR;

}
return retCode;
}
}
```

Message severity

INFORMATION = 1 = Info message only.

WARNING = 2 = Warning (OK/Cancel)

ERROR = 3 = Error (Error but continue validation to find more error)

FATAL = 4 = Fatal (Error and stop validation)

Message methods

- addObjectMessage (String objectId, String msgId, short severityCode): Return a message with the severity code and set focus on the object Id specified when user clicks on the message link on the UI. ObjectID is the object ID specified in the Designer. Message ID identifies the message stored in the message table in the database.

  Attention: See Javadoc for additional overloading addObjectMessage methods that passes additional parameters.

- addLineMessage (String msgId, short sevType): Same as addObjectMessage but return focus to the entire line when user clicks on the message link on the UI
- addRSMessage (String msgId, short sevType): Same as addObjectMessage but return focus to the entire result set when user clicks on the message link on the UI

Example: Looping through result set

```
private void loopMyResultSet(ResultSetInterface rsI) throws DEException {

RowSetInterface roI = rsI.getRowSet();

// loop and find rows that are new but exclude those marked deleted

RsIterator rsT = rsI.findInit(roI.ROW_New, roI.ROW_MarkDeleted, true)

while (rsT.next()!=roI.UNDEFINED_CONTEXT) {
```

```
(do your logic here)

}

}
```

system.applicationinterface.RowSetInterface

This interface provides methods to get and put data (columns) onto a row, inspect row flags (new, updated, markdeleted, etc), add rows to result set, get original value selected for an Object Id, and so on.

Here are some commonly used methods provided by the interface (see javadoc for more explanations)

- getStringValue(String sColumnName): Return the value in the column specified with Object ID. If the value is blank or null in the database, an empty string will be returned (not a null).
- setStringValue(String value,String sObjectId): Set a value into the column specified with Object ID

Example: Getting and setting data with RowSetInterface

```
private void loadSettings (ResultSetInterface rsI) throws DEException {`

  RowSetInterface roI = rsI.getRowSet();
  /* get a value */
  String sMethod = roi.getStringValue("S_PO_AUTO_NUM_TYPE");
  /* set a value */
  if (sMethod.trim().equals(""))

    roi.setStringValue("S","S_PO_AUTO_NUM_TYPE");

}
}
```

system.applicationinterface.AppInterface

Provide methods to get various global settings independent of the result set you are in such as current App Id, System Name, Language No and User Id.

AppInterface is obtained via the ResultSetInterface's getApplication method.

Here are some commonly used methods provided by the interface (see javadoc for more explanations)

- **getSystemName()**: Return system name selected by user when login.
- **getUserId()**: Return ID of login user.
- **getConstant(String constId)**: Return the object containing the application constant identified in the ID

Example: Getting User Id with AppInterface

```
private String getUserId (ResultSetInterface rsI) throws DEException {

AppInterface appI = rsI.getApplication();

String sUserId = appI.getUserId();

return sUserId;

}
```

Example: Getting the current application ID

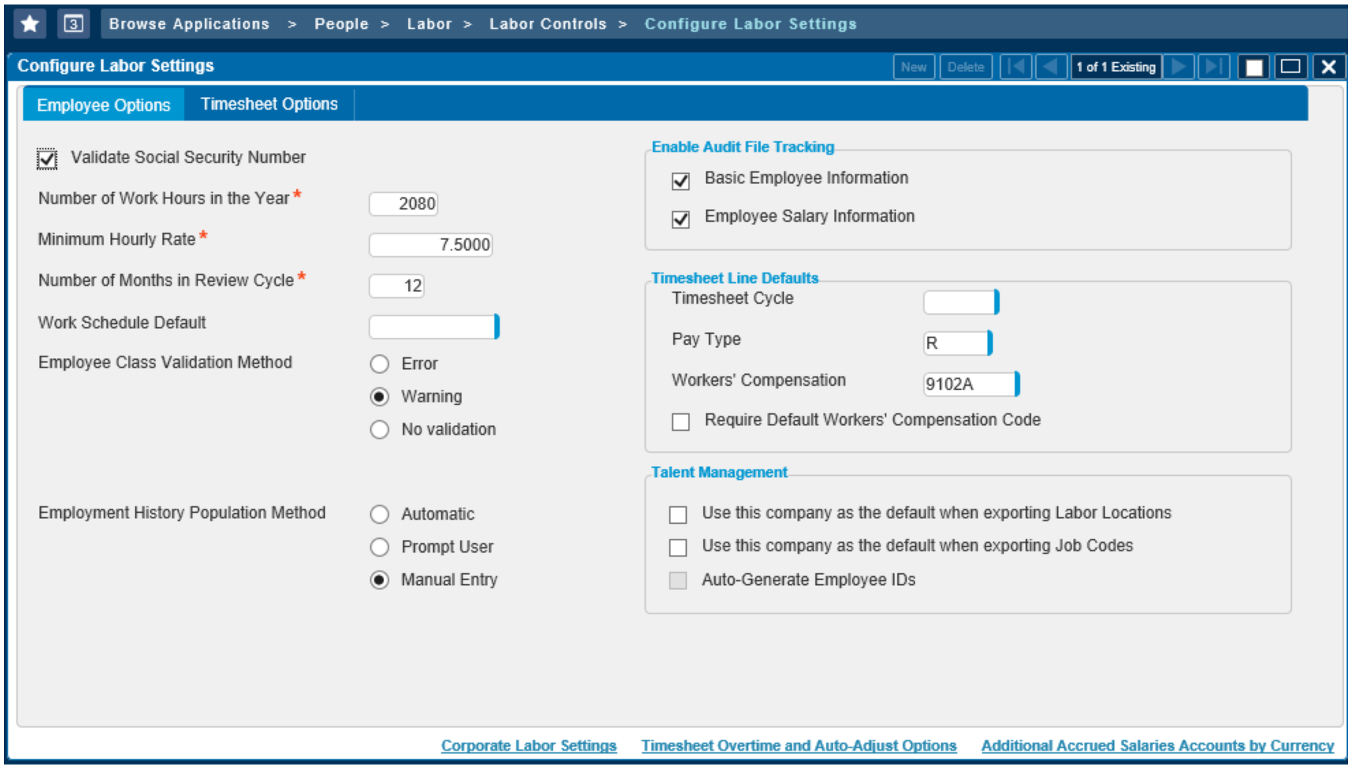
```
private String getAppId (ResultSetInterface rsI) throws DEException {

AppInterface appI = rsI.getApplication();

String appId = appI.getAppName();

return appId;

}
```

Global Constants

The framework stores data that are set in setting tables (such as GL Setting, PO Setting, and so on) as global constant. Data is retrieved once by the framework from the database and stored in static hashtables by framework.

Application does not have to use SQL to get the value. Application can simply get the value via the getConstant method.

Example: Getting the Validate Social Security Number Flag in Labor Settings



[✓] Validate Social Security Number

Number of Work Hours in the Year* 2080

Minimum Hourly Rate* 7.5000

Number of Months in Review Cycle* 12

Work Schedule Default

Employee Class Validation Method: Error [✓ Warning] No validation

Employment History Population Method: Automatic Prompt User [✓ Manual Entry]

Enable Audit File Tracking

[✓] Basic Employee Information

[✓] Employee Salary Information

Timesheet Line Defaults

Timesheet Cycle

Pay Type R

Workers' Compensation 9102A

[ ] Require Default Workers' Compensation Code

Talent Management

[ ] Use this company as the default when exporting Labor Locations

[ ] Use this company as the default when exporting Job Codes

[ ] Auto-Generate Employee IDs

Corporate Labor Settings Timesheet Overtime and Auto-Adjust Options Additional Accrued Salaries Accounts by Currency

```
private String getLabSettingSSNFl (ResultSetInterface rsI) throws DEException {

AppInterface appI = rsI.getApplication();

String valSSNFl = (String) appI.getConstant("CP_LABSETTINGS_SSNVALFL");

return valSSNFl;
}
```

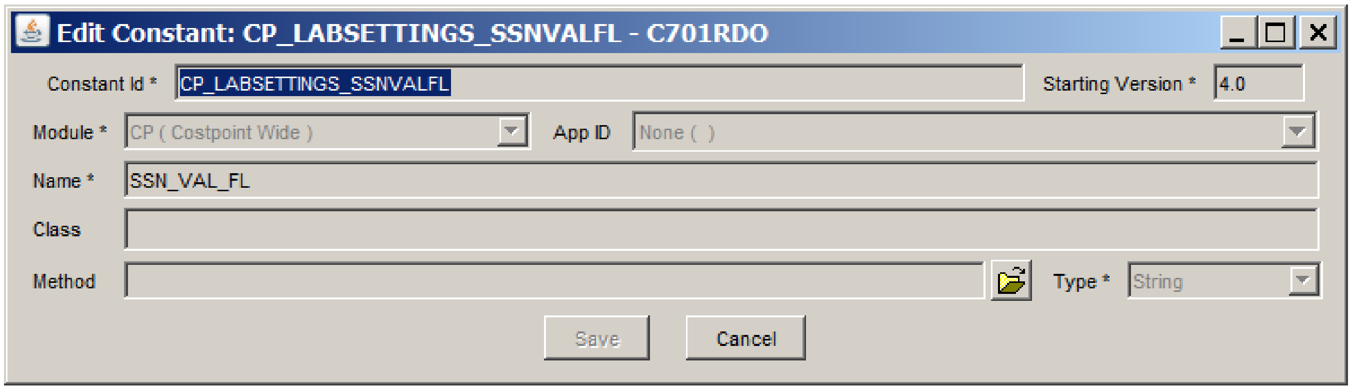
Example - Get Project Control's top level length

```
private short validateSomeObjects (ResultSetInterface rsI) throws DEException {

// get app interface from ResultsetInterface

AppInterface api = rsI.getApplication();

// get top level len

Integer nTopLvlLenNo = (Integer) api.getConstant("CP_PROJCNTL_TOPLVLLENNO");
}
```

Deltek

List of Global Constants

In the Designer, go to Constant and filter with ID starting with CP only. Open the constant. Any constant that does not have an entry in the Class and Method fields is a global constant.



Constant Id *	CP_LABSETTINGS_SSNVALFL	Starting Version *	4.0
Module *	CP (Costpoint Wide)	App ID	None ( )
Name *	SSN_VAL_FL		
Class			
Method		Type *	String

Using Constants as Variable in the Designer

To use these constants as variable in the Designer, you must assign the constant to the application in the Designer. Then you can use it as part of the formula for Editable, Visible, Value, and so on. You can also use it as part of the text for labels or status text.

Automatic Constant

Automatic constant is a subset of global constant that is always available for use. You do not have to assign them to your application. These are:

- CP_USER_ID = ID of the user who currently logs in.
- CP_LANG_NO = Language No being used by the current login session.
- CP_COMPANY_ID = Company the user is working on in the current login session.
- CP_APP_ID = App ID the user is working on in the current login session.
- CP_CURRENT_DATE = Date time of the server when user logins to an application
- CP_SESSION_ID = Station ID assigned to the user when login to Costpoint Web.
- CP_GLCONFIG_REFSTRUC1LBL = Ref1 Label
- CP_GLCONFIG_REFSTRUC2LBL = Ref2 Label
- CP_WUSERCOMPANY_SUPPRESSCSTFL = User Company Cost Suppression Flag
- CP_WUSERCOMPANY_SUPPRESSLABFL = User Company Labor Suppression Flag
- CP_WUSERCOMPANY_SUPPRESSPRCFL = User Company Price Suppresion Flag
- CP_WF_KEY
- CP_WF_CASE_KEY
- CP_WF_ACTIVITY_KEY

- CP_WF_OPTIONAL_KEY

Creating your own application constants

In addition to the global constants, you can create your own constants via your own java class. Once it is created, you can use it in your application by assigning it to your app in the Designer. It will be then available to your server code as well as client code (formula, label, status text set in Designer).

The object returned can be String object, or a Date, Integer or Double.

If you return a String containing a date, then you must format the string with the system date format (yyyy-MM-dd). Using other format will interfere with the system trying to reformat the date to the correct locale of the user. Generally, we suggest that you return a Date instead. The system will automatically convert the date to the proper format for presentation and you do not need to know what format to use if you do it yourself. In addition, you can also use the Date object at the server for other purposes.

To create your own application constant:

1. In the Designer: Create a new constant Id. Specify the java class and method used to populate the value of this constant. Then assign constant to your application

2. Create the java class and method to return the value. Class can be an existing class in your extensions package.

3. The java method must have this signature

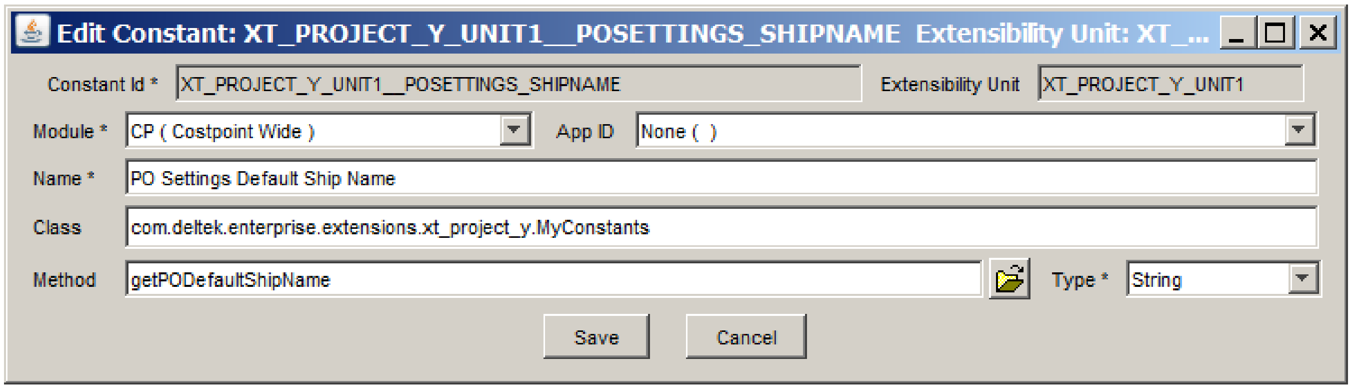
   public Object xxxx (AppInterface appl) throws DEException {}

4. Use the AppInterface.getSqlManager method to get an instance of SqlManager.

Example: Create constant Ship Name from the Default Ship ID in the PO_SETTINGS

Designer:

Deltek



Edit Constant: XT_PROJECT_Y_UNIT1__POSETTINGS_SHIPNAME Extensibility Unit: XT_...

Constant Id *	XT_PROJECT_Y_UNIT1__POSETTINGS_SHIPNAME	Extensibility Unit	XT_PROJECT_Y_UNIT1
Module *	CP (Costpoint Wide)	App ID	None ( )
Name *	PO Settings Default Ship Name		
Class	com.deltek.enterprise.extensions.xt_project_y.MyConstants		
Method	getPODefaultShipName	Type *	String

Java class:

```
package com.deltek.enterprise.extensions.xt_project_y;
import java.sql.*;
import com.deltek.enterprise.system.applicationinterface.*;
public class MyConstants {
public String defaultShipId;
public String defaultShipName;
public Object getPODefaultShipName (AppInterface appI) throws DEException, SQLExc
eption {

  SqlManager sqlMgr = appI.getSqlManager("DATA",this);
  String sysName = appI. getSystemName();
  /* Get global constant Default Ship ID */
  defaultShipId = (String) appI.getConstant("CP_POSETTINGS_DFLTSHIPID");
  /* Select ship name from ship id */
  String Select = "SELECT SHIP_NAME FROM SHIP_ID " +

"WHERE SHIP_ID = :defaultShipId INTO :defaultShipName";`

  sqlMgr.SqlExecuteQuery(Select);
  }
}
```

system.applicationinterface.LoggerInterface

All logging in Java code should be done using the provided LoggerInterface. Avoid using System.out. Use LoggerInterface to enable more flexibility and extensibility in logging information. The advantages of using the Logger are:

- It is possible to print to the console or some log files by changing properties in the config file.

- No need to comment/uncomment System.out.println in the code
- The logging messages from a particular package or a particular dir can be filtered for analysis/debugging purpose

How to Add Logging in Your Class:

- Import com.deltek.enterprise.system.applicationinterface.LoggerInterface
- Get a LoggerInterface from AppInterface
- Use the logging methods from the LoggerInterface.

Example: Using the LoggerInterface

```
import com.deltek.enterprise.system.applicationinterface.LoggerInterface;
import com.deltek.enterprise.system.applicationinterface.DEException;
import com.deltek.enterprise.system.applicationinterface.ResultSetInterface;
import com.deltek.enterprise.system.applicationinterface.AppInterface**;**
public class myClass {
public short validateMyEntity (ResultSetInterface rsI) throws DEException {


  AppInterface appI = rsI.getApplication();
  **LoggerInterface logger = appI.getAppLogger();**
  logger**.debug**("This a debug message");
  logger**.info**("This is a info message");
  logger.**warn**("This is a warning message");
  logger.**error**("This is an error message");
  logger.**fatal**("This is a fatal message");
}
}
```

CPlog4j.properties

At run time, the system will read the default configuration file CPlog4j.properties located in the classpath (currently located at \applications\enterprise\properties). This file contains the specified output, severity level and layout of messages.

Severity levels are (from lowest to highest): Debug, Info, Warn, Error, Fatal. The severity level set in the configuration file indicates the lowest to be output. For example, if the level is set at Warn, then only Warn, Error and Fatal level messages are output.

Deltek

system.applicationinterface.CPConstants

The CPConstants interface contains constants that are used system wide. These are java constants. Do not confuse this with Costpoint global constants that were discussed in the AppInterface section.

For example, CP_OK, which corresponds to 0, can be returned from the validation when no errors are encountered. Other constants in this interface include CP_ACCT_DELIM, CP_MAX_PROJ_LVLS, and CP_MAX_NUM_SUB_PDS.

To use CPConstants, import com.deltek.enterprise.system.applicationinterface.CPConstants. If your class implements CPConstants, you can use the constants such as CP_OK without the qualifier. Otherwise, you need to use them with the interface CPConstants as the qualifier, for example, CPConstants.CP_OK

Example: Class using CPConstants

```
import com.deltek.enterprise. system.applicationinterface.CPConstants;
public class PjmbasicMyClass {


  public short retcode = CPConstants.CP_OK;
}
```

system.applicationinterface.SqlManager

SqlManager class is used to perform SQL operation on the database. You request an instance of SqlManager from ResultSetInterface (or AppInterface or ActionInterface). With it, you can use its methods to execute SQL statement.

Rules

- Avoid excessive request of SqlManager instances. For example, do not request SqlManager in a loop. Request one instance and then use it in the loop. When writing common validation method, use a passed in SqlManager as a parameter instead of creating a new SqlManager.
  The system will throw an application error if more than 20 SqlManager instances are requested within one single plug-in call (an application method called by the system for a single event). This includes all SqlManagers requested from functions invoked from the plug-in class.
- Never add table owner in front of table name. Owner is assumed by the name setup in the server connection pool.
- DbF functions are provided to cover cross DBMS SQL syntax. Use them in SQL statement like a SQL function (see an example below).

Some commonly used methods in SqlManager are:

- **SqlPrepareAndExecute(String)**: prepares and executes SQL statement.
- **boolean SqlFetchNext()**: fetches next row from result set.
- **SqlExecSP(sExpectedVersion,bCheckOnly,sSPName,sParams,bCommit)**: invokes stored procedure.
- **SqlGetModifiedRows**: Get the number of rows affected by the last SQL statement.
- **boolean SqlExecuteQuery(String)**: executes select statement and fetches first row.

Example: SqlExecuteQuery using class variables for bind and into variables

```
/* Bind variables are declared as class variables. SqlManager uses reflection to
read and write values back to your object. All bind variables are preceded with t
he colon (:) in the SQL statement.
If your class has been extended and the SQL is being done on the subclass, do not
use class variables as bind variables. Use the local hashmap approach (see next e
xample)
*/
public String sShipId;
public String sShipName;
public void posettingsGetShipDesc (ResultSetInterface rsI) throws SQLException, D
EException {
SqlManager sqm = rsI.getSqlManager(this);
RowSetInterface roI = rsI.getRowSet();
sShipId = roI.getStringValue("PO_SETTINGS___DFLT_SHIP_ID");
String sSelect = new StringBuffer()
.append("SELECT SHIP_NAME FROM SHIP_ID WHERE SHIP_ID = :sShipId")
.append(" INTO :sShipName").toString();
sqm.SqlExecuteQuery(sSelect);
roI.setStringValue(sShipName,"SHIP_NAME");
}
```

Example: SqlExecuteQuery using a local hashmap to hold bind/ into variables

```
public void posettingsGetShipDesc (ResultSetInterface rsI) throws SQLException, D
EException {
SqlManager sqm = rsI.getSqlManager(this);
RowSetInterface roI = rsI.getRowSet();
String sShipId = roI.getStringValue("PO_SETTINGS___DFLT_SHIP_ID");
String sShipName = null;
// put variables into a local HashMap
HashMap bindHash = new HashMap();
bindHash.put("sShipId", sShipId);
bindHash.put("sShipName", sShipName);
// Put HashMap in SqlManager
sqlMgr.setBindClasses(bindHash);
String sSelect = new StringBuffer()
.append("SELECT SHIP_NAME FROM SHIP_ID WHERE SHIP_ID = :sShipId")
.append(" INTO :sShipName").toString();
sqm.SqlExecuteQuery(sSelect);
// Get the value out of hashmap
HashMap outHash = sqlMgr.getBindsTable();
sShipName = (String) outHash.get("sShipName");
roI.setStringValue(sShipName,"SHIP_NAME");
}
```

Using DbF functions

If your customer runs Costpoint on multiple DBMS platforms, your SQL statement must work for all platform. DbF functions are provided to avoid different syntax for different DBMS platform. For example, outer join syntax is different for SQL Server versus Oracle. Costpoint provides the DbF_OuterJoin function to avoid having to write different SQL. Alternatively, you can write SQL in ANSI syntax which would work for all platform.

DbF function is used like a SQL function (not a java function). Therefore, do not concatenate your SQL statement with DbF function java calls. Include such function inside the SQL statement. The system will parse the DbF function inside the statement to the appropriate back end syntax.

For a complete list of DbF functions, open the Sql Editor in the Designer and select the drop down box for DbF functions.

Example: Using DbF_Outerjoin function

DbF_Outerjoin automatically converts to ANSI outerjoin syntax after the system parses the statement.

`String select = "SELECT T1.USER_ID, T1.EMPL_ID, T2.LAST_NAME FROM USER_ID T1, EMPL T2 WHERE DbF_OuterJoin(T1.EMPL_ID,T2.EMPL_ID)"`

Example: Using DbF_OptionalString to insert NULL when field is empty

SqlManager will insert a blank string if the value passed in is empty. Use DbF_OptionalString for column that needs to be NULL if the value is blank.

```
String sInsert = "INSERT INTO PO_HDR_DFLT (PO_ID, PO_RLSE_NO,RQ_ID,SHIP_I
D,......) " +
" VALUES (:sPoId, :nPoRlseNo, DbF_OptionalString(:DfltShipId), .....)
sqlMgr.SqlPrepareAndExecute(sInsert);
```

Using other key words

- Key word CP_USER_ID can be used to substitute the value of the User Id in SQL statement. Use it for MODIFIED_BY column (see example below)
- Key word CP_COMPANY_ID can be used to substitute the value of the current company id the user is working with in the current session.
- Key word CP_APP_ID can be used to substitute the value of the current app id the user is working with in the current session.
- Key word CP_CURRENT_DATE can be used to substitute the value of the current database server date and time. Alternatively, you can also use the DbF_DateCurrent function.

Example: Using key word: CP_USER_ID and DbF functions

```
sProjId = rowsetI.getStringValue("PROJ_ID");
sOldOrgId = rowsetI.getStringValue("ORG_HIST___OLD_ORG_ID");
String sSql = new StringBuffer()

  .append("UPDATE PROJ SET ORG_ID = :sOldOrgId,")
  .append("MODIFIED_BY = :CP_USER_ID,TIME_STAMP = DbF_DateCurrent()")
  .append(" WHERE PROJ_ID LIKE :sProjId DbF_ConcatChar() '%'").toString();

sqlMgr.SqlPrepareAndExecute(sSql);
```

Example: Checking for concurrency

Concurrency error should be captured as a FATAL error.

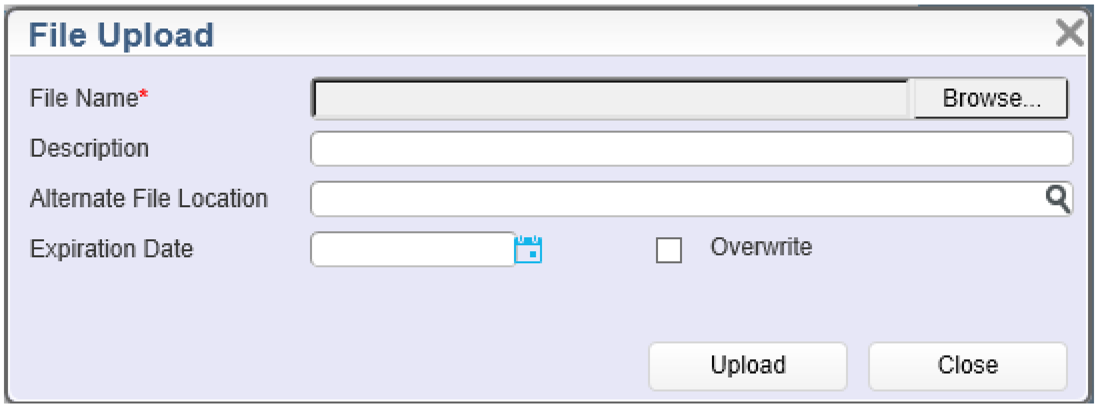
```
sqlMgr.SqlExecute();
if (sqlMgr.SqlGetModifiedRows() == 0)
{ // NO row updated by last UPDATE statement

  String tableName = "xxxxx";
  String dbOperation = "xxxx"

String[] msg = { tableName, dbOperation }; //table name, operation
rsI.addRSMessage("XT_PROJECT_Y_NO_ROWS_UPDATED",rsI.INFO, msg);
return rsI.INFO;
}
```

system.applicationinterface.FileHandlerInterface

File Uploading

For files that must be processed by Costpoint (for example, preprocessor input files), the user can upload the file using the File Upload Manager via the Process->File Upload option on the menu. This uploads the file to the Costpoint database table W_USER_FILE_CATLG. Files can also be placed in an alternate file location (see next section).

File Uploading UI



A dialog appears when clicking on the File Upload option on the menu bar.

- Enter the file name or Browse to the file (located in your drive or a network drive).
- Enter a brief description.
- Enter the expiration date. A purge process can be scheduled to remove these files from the database based on the expiration date.
- Check Overwrite box if you want to overwrite a previous version. (Otherwise, an error message may be returned if a duplicate file is found).

The file will be saved in the table W_USER_FILE_CATLG with the same file name. The following are stored as key for this table: User ID, Company ID, App ID and file name

Alternate location

To process a file, the file must either be uploaded in the database or uploaded in an alternate location (disk file).

An alternate location is a folder where the application server can access/read/write disk files. Once the folders are created on disk, you associate the URL with a logical name inside Costpoint. An alternate file location is specified in the Manage Alternate File Locations application. After a file is placed in that location, it can be used in the application if the application screen provides input fields for entering file location ID and file name.

FileHandlerInterface

To access files already uploaded for processing, use the FileHandlerInterface in com.deltek.enterprise.system.applicationinterface package.

Common methods in FileHandlerInterface include:

- boolean **fileOpen**(java.lang.String fileName, int nStyle): Open a DB file uploaded in database
- boolean **fileOpen**(java.lang.String fileName, int nStyle, String altLocation): Open a file uploaded in alternate location
- String **fileRead**(int len): Read a buffer of characters from a file to a string starting with the current position to the new position (current plus len passed in).
- String getGetStr():Read the next line as String from an opened file.
- void **fileClose** (): Close an opened file.

Example: Copying a file uploaded in DB to a new file in DB

```
import com.deltek.enterprise.system.applicationinterface.FileHandlerInterface;
public class TFileMgr {
public short copyFile (ActionInterface aci) throws DEException {
FileHandlerInterface fi = aci.getFileManager();
ResultSetInterface rsi = aci.getResultSet();
RowSetInterface roi = rsi.getRowSet();
String fileName = roi.getStringValue("MY_RS_FILE_NAME");
fi.fileCopy(fileName,fileName+"_copy",true);
return 0;
}
}
```

Example: Reading a file uploaded in DB

```
import com.deltek.enterprise.system.applicationinterface.FileHandlerInterface;
public class TFileMgr {
public short copyFile (ActionInterface aci) throws DEException {
FileHandlerInterface fi = aci.getFileManager();
ResultSetInterface rsi = aci.getResultSet();
RowSetInterface roi = rsi.getRowSet();
String fileName = roi.getStringValue("MY_RS_FILE_NAME");

  if (!fi.fileOpen(fileName,fi.OF_Read)) {
        rsi.addMessage(....);
        return rsi.ERROR;

}

  String tempStr = null;
  /* read one line at a time */

while ((tempStr = fi.fileGetStr()) != null) {
... process data in tempStr...
}


  fi.fileClose();
  return 0;
}
}
```

Example: Copying a file in db and write out to a disk file in alternate location

```
public void copyFileToAltLocation(ActionInterface action) throws DEException {
RowSetInterface row = action.getResultSet().getRowSet();
String altLocation = row.getStringValue("ALT_LOCATION");
String fileName = row.getStringValue("FILE_NAME");
String newFileName = row.getStringValue("ALT_FILE_NAME");
// open file
FileHandlerInterface fileHandler = action.getFileManager();
if (**fileHandler.fileOpen**(fileName, fileHandler.OF_Read)) {
String content = fileHandler.fileRead(fileHandler.length());
**fileHandler.fileClose**();
// create new file
newFileName = (newFileName == null) ? fileName : newFileName;
if (fileHandler.fileOpen(newFileName, fileHandler.OF_Create, altLocation)) {
**fileHandler.fileWrite(content);**
**fileHandler.fileClose();**
// Ok
String[] s = {newFileName};
action.addMessage("XT_PROJECT_Y_FILE_CREATE_OK", action.INFORMATION, s);
}
else {
// cannot create new file
String[] s = {newFileName};
action.addMessage("XT_PROJECT_Y_FILE_CREATE_FAILED", action.ERROR, s);
}
}
else {
// existing file is not found
String[] s = {fileName};
action.addMessage("XT_PROJECT_Y_FILE_NOT_FOUND", action.ERROR, s);
}
}
```

In addition to system's applicateinterface package, application can also import and use classes in the system's utils package.

system.utils.Numbers

Provides methods to do rounding, compare, truncate, and so on.

Here are some commonly used methods provided by the class.

- public static int compare(double number1, double number2): returns 1 if number1 > number2, 0 if number1=number2, else return -1
  This method will help avoid errors caused by floating point arithmetic and will allow reliable comparison of two double values
  For example, the number in regular, decimal, format 123.45 is stored as 1.2345E02; one is stored as 1.0E01.
  This means that java uses quite different rules (floating point arithmetic) to do any arithmetic operation.
  Rounding errors can happen when you use two or more double variables or just add some number to the double variable.
  For example:
  - double d1=1.1233357
  - double d2=4.165
  - double d=d1-d2
  - double d3=d1-d2-d
  We would expect d3 to equal zero. However, d3 is actually = 2.220446049250313E-16.
  Use this method compare instead of doing your own arithmetic for comparison.
- public static double round(double value, int scale)
  Round a number to the scale specified.

System.utils.UIFormat

Provide methods to format number or date to string for presentation purposes.

Here are some commonly used methods provided by the class.

- public static String numberToStr(AppInterface appI, double amount, int nScale, boolean bThousSep,boolean bPadDec)
- public static String dateTimeToStr(AppInterface appI,Calendar datetime, boolean bDate,boolean bTime, boolean bFourDigitYear)

Example: Formatting a currency amount in an error message

```
import com.deltek.enterprise.system.applicationinterface.*;
**import com.deltek.enterprise.system.utils.UIFormat;**
public class MyRowValidation {
public short validateRow (ResultSetInterface rsi) throws DEException {
RowSetInterface roi = rsi.getRowSet();
double parentAmt = roi.getdouble("RQ_LN_TOT_CALC_AMT");
double childTotAmt = roi.getdouble("RQ_LN_TOT_AMT");
if (parentChangeAmt != childChangeAmt) {
AppInterface app = rsi.getApplication();
String parAmt = **UIFormat.currencyToStr**(app, parentAmt,true,true,true);
String childAmt = **UIFormat.currencyToStr**(app, childAmt,true,true,true);
rsi.addObjectMessage("TOT_AMT_OBJ_ID","XT_PROJECT_Y_AMT_INVALID",rsi.ERROR,

      `new String[] {parAmt,childAmt});`

return rsi.ERROR;
}
return 0;
}
```

Plug-in Classes Are Stateless

Application plug-in classes invoked by the framework (such as object/cell/line validations, actions, etc) are stateless objects. Between method calls, class variables can be of any state since the framework does not clear its cache. Therefore, class variables should be initialized inside the method when the method is called as a plug-in entrance.

For example: You have two object validation methods implemented in the same class. These two methods will be invoked independently. Do not set class variables in one method that will be used for the other method.

Note: If one method in the plug-in class is invoking other private method in the same class, no re-initialization is necessary in the other private methods since this is still within the plug-in lifetime. Initialization should happen only at the beginning of those public methods, which are called by the framework.

Let's review RS related events. List of RS events available to extensibility developer may depend on if you are creating a brand new RS or Extending Existing RS created by regular Costpoint developer, but timing of those events is similar in nature. For example if you are extending existing RS - event available to you will be After RS Open, which will be invoked after regular RS Open event. You can guess this by the name of the event. But if you are creating a brand new RS -- the event will be named RS Open. Similar picture with other events.

RS Open Event

RS Open Event

- RS Open event happens when the framework starts to open a result set. Typically, developers are performing various licensing or setup related checks and provide an error if something is missing that will prevent the normal behavior of a given application. You can also perform various RS setup related calls associated with dynamic Result Sets or Dash Parts RS.

RS Population Event

RS Populate Event

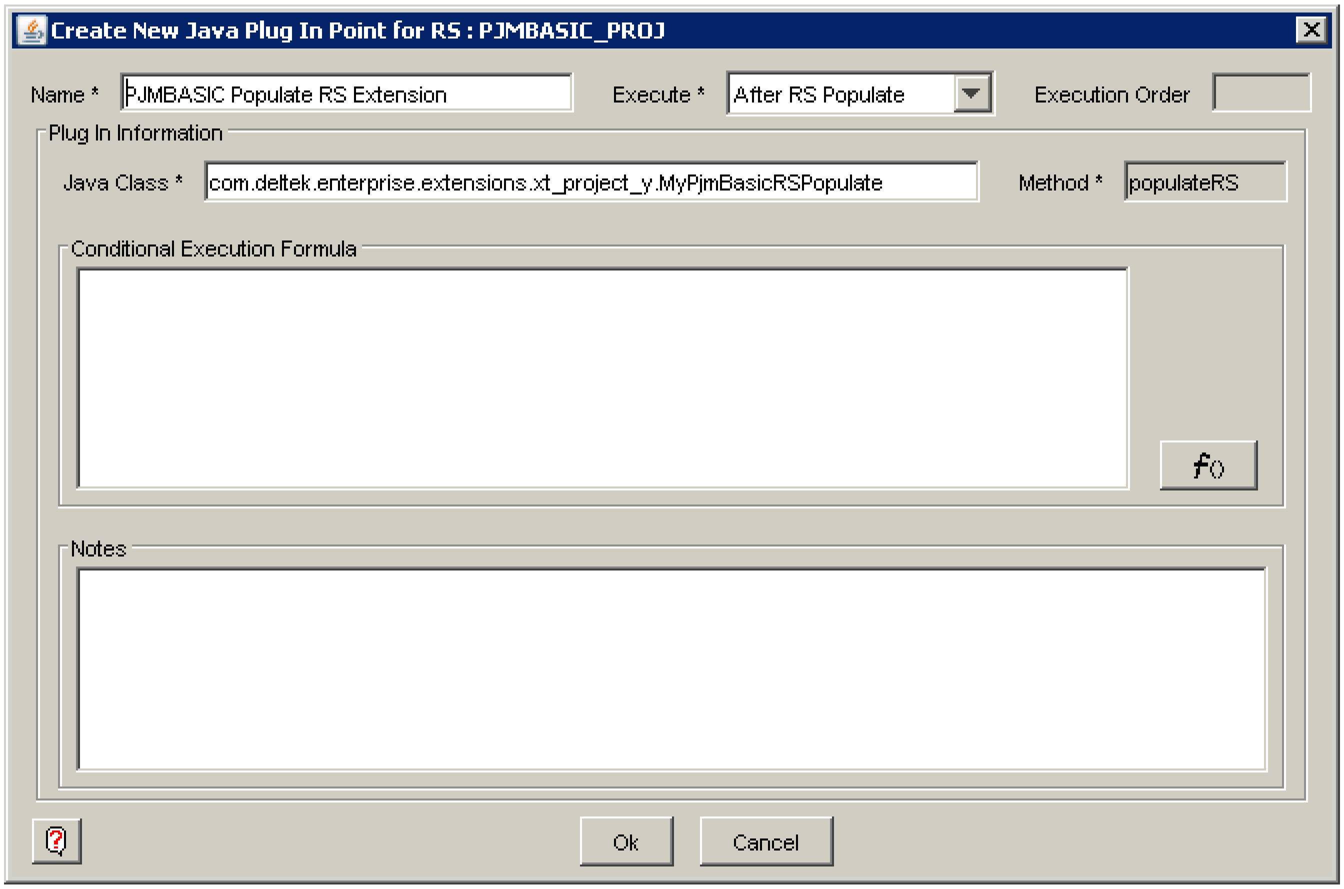
- RS Populate event happens when the framework executes the data retrieval for a result set. This can be initiated by user selects Query on the result set or the result set auto populates its data on opening.
- On this event, the framework will select the data based on the SQL statement entered in the Designer for result set. If you register a class for the RS Populate event, the class will be called by the framework before the data is sent to the client.
- RS Populate class is used to manipulate the data beyond the Select statement. For example, getting additional data (such as description or name from another table) where additional table join in the select SQL is costly for performance or not possible. It is also used to populate or add rows to the result set that are not coming from the database.

Caching Scheme

The framework is optimized to retrieve the rows from the database only as needed. It uses cache and buffering mechanism to retrieve only enough data to fill the UI screen. Each time the system needs to fetch additional rows (on user scrolling at the client), it will go to the database and fetch more. The size is indicated in DB_FETCH_SIZE setting in enterprise.properties. As user scrolls for more data, the server will check its cache before going to the database to get more.

The cache size is indicated by the RS_CACHE_SIZE setting. Once the rows reached the RS_CACHE_SIZE, old rows will be discarded to make room for new rows. New rows entered by user or rows that have been edited are stored separately and not included in this cache size limit.

Due to this scheme, RS populate plug-in class may be called more than one time. So do not use class variables in the method unless they are initialized each time.


Create New Java Plug In Point for RS : PJMBASIC_PROJ

Name * PJMBASIC Populate RS Extension
Execute * After RS Populate
Execution Order

Plug In Information

Java Class * com.deltek.enterprise.extensions.xt_project_y.MyPjmBasicRSPopulate
Method * populateRS

Conditional Execution Formula

Notes

- Enter a descriptive name for the plug-in. Select After RS Populate. Your class will be executed after the standard Deltek RS Populate plug-in.
- Enter the fully qualified package and class name. Your java class must implement PopulateRSInterface interface and the method populateRS.
- Use method nextBufferRow() from the ResultSetInterface to loop through rows retrieved by the framework. The system sets context automatically so you do not need to call method setRowSetContext in the loop.

Example: Looping through rows selected by framework and set field values

```
package com.deltek.enterprise.extensions.xt_project_y;
import com.deltek.enterprise.system.applicationinterface.*;
public class MyPjmBasicRSPopulate implements PopulateRSInterface {
public void populateRS(ResultSetInterface rsI) throws DEException {
RowSetInterface rowSetI = rsI.getRowSet();
int i;
while ( (i= rsI.nextBufferRow())!=rowSetI.UNDEFINED_CONTEXT){ {
rowSetI.setStringValue("","CURRENT_FISCALYEAR");
if (rowSetI.getStringValue("REV_LBL").equals("N/A")) {
rowSetI.setStringValue(null,"REV_LBL");
rowSetI.setDouble(null,"REV_AMT");
}
}
}
}
```

Validation event

Types of Validation Events

Validations are plug-in events fired by the framework so the application will have a chance to validate the data.

There are three types of validations: Object validation, Line (or Row) Validation, and RS Validation. The timing of the events at the server depends on the Validate Frequency mode on login.

Rowset definition

A rowset is a group of rows within a result set having the same parent row. Thus, top-level result set in an application will always contain only one rowset. A non top-level result set can have multiple rowset if there are multiple parent rows.

For example:

Two voucher headers A and B each having two voucher lines: A1, A2 and B1, B2 respectively. In such example, there are two result sets: Voucher Header and Voucher Line. Since Voucher Header is a top-level result set, there is only one rowset with two rows A and B. For the Voucher Line result set, there are two rowsets: A1 and A2 with parent A, and B1 & B2 with parent B.

This explanation will help you understand the discussion in the rest of this chapter.

Object Validation

Object Validation should be used to validate a single object.

Timing

In Field mode, Object Validation is invoked when value in the object has changed and user tabs out of it. Once validation is done, it is not called again when user just tabs through it again unless the value is changed again. If the field's value is changed programmatically by the application java code, no Object Validation is fired.

In Line mode, Object Validation is delayed and called when user leaves the current line to another line and the object value has been changed. So while user keep changing the value but still stay on the same line, it is not yet called.

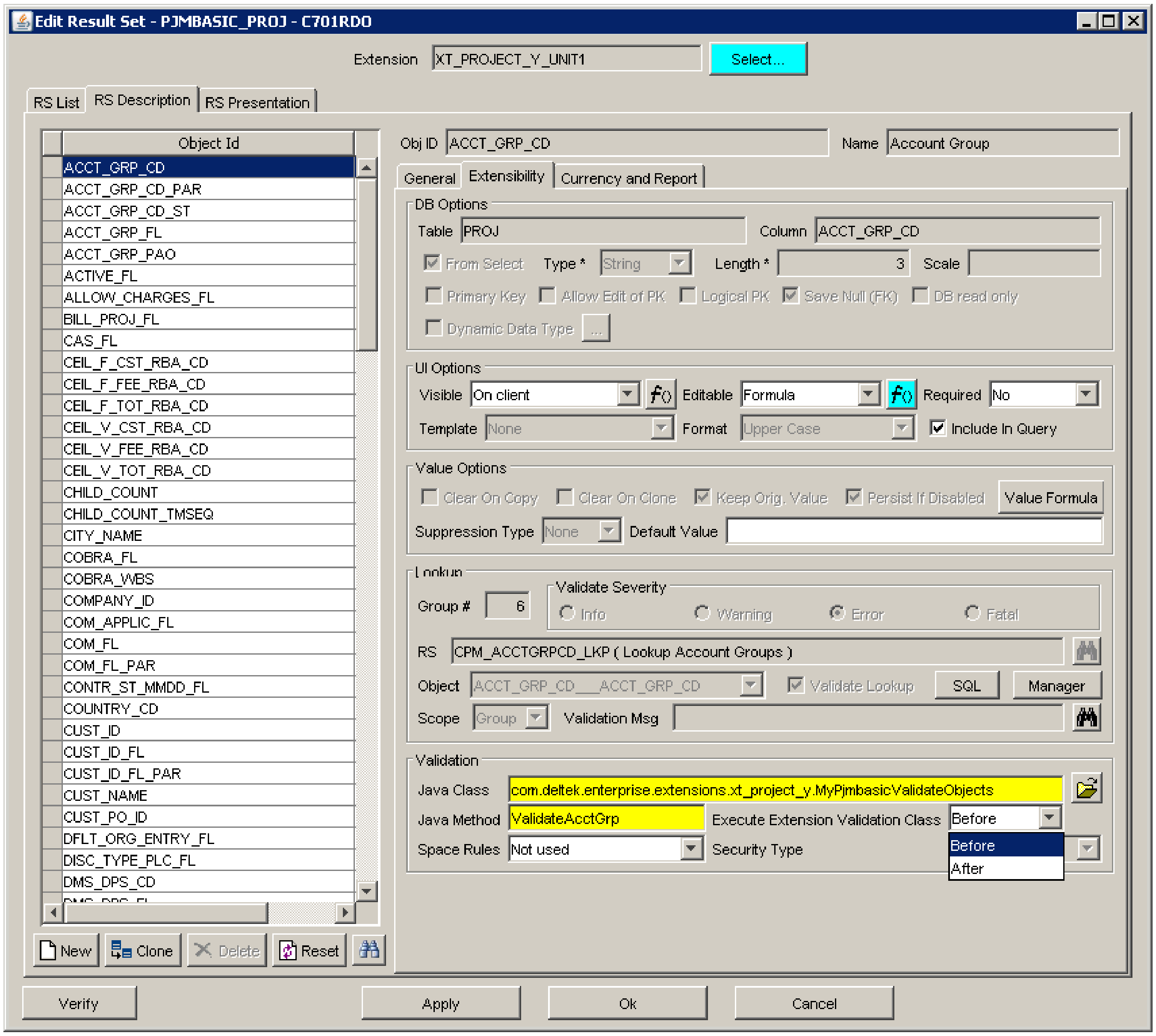
In Application mode or web service mode, Object Validation is only called on Save.

Rules

- Object validation is never invoked if the value is changed to a blank or null. It is only invoked if the value has been changed to a non-blank value. Therefore, you should never have Object Validation code that checks if the field is empty because the plug-in code would never be executed. If you need to check that the field is empty to do something, do this check in the Row Validation plug-in class.
- Object validation should be written to look at the value of a single object only. If the validation depends on the value of other objects on the screen, it must be done in Row validation or RS validation when user has completed the input for the dependent fields.

Deltek

Extensibility


Edit Result Set - PJMBASIC_PROJ - C701RDO

Extension: XT_PROJECT_Y_UNIT1 — Select...

RS List | RS Description | RS Presentation

Object Id
ACCT_GRP_CD
ACCT_GRP_CD_PAR
ACCT_GRP_CD_ST
ACCT_GRP_FL
ACCT_GRP_PAO
ACTIVE_FL
ALLOW_CHARGES_FL
BILL_PROJ_FL
CAS_FL
CEIL_F_CST_RBA_CD
CEIL_F_FEE_RBA_CD
CEIL_F_TOT_RBA_CD
CEIL_V_CST_RBA_CD
CEIL_V_FEE_RBA_CD
CEIL_V_TOT_RBA_CD
CHILD_COUNT
CHILD_COUNT_TMSEQ
CITY_NAME
COBRA_FL
COBRA_WBS
COMPANY_ID
COM_APPLIC_FL
COM_FL
COM_FL_PAR
CONTR_ST_MMDD_FL
COUNTRY_CD
CUST_ID
CUST_ID_FL
CUST_ID_FL_PAR
CUST_NAME
CUST_PO_ID
DFLT_ORG_ENTRY_FL
DISC_TYPE_PLC_FL
DMS_DPS_CD

Obj ID: ACCT_GRP_CD — Name: Account Group

General | Extensibility | Currency and Report

DB Options
Table: PROJ — Column: ACCT_GRP_CD
[✓] From Select — Type *: String — Length *: 3 — Scale:
[ ] Primary Key [ ] Allow Edit of PK [ ] Logical PK [✓] Save Null (FK) [ ] DB read only
[ ] Dynamic Data Type

UI Options
Visible: On client — Editable: Formula — Required: No
Template: None — Format: Upper Case — [✓] Include In Query

Value Options
[ ] Clear On Copy [ ] Clear On Clone [✓] Keep Orig. Value [✓] Persist If Disabled — Value Formula
Suppression Type: None — Default Value:

Lookup
Group #: 6
Validate Severity: Info, Warning, [✓ Error], Fatal
RS: CPM_ACCTGRPCD_LKP (Lookup Account Groups)
Object: ACCT_GRP_CD___ACCT_GRP_CD — [✓] Validate Lookup — SQL — Manager
Scope: Group — Validation Msg:

Validation
Java Class: com.deltek.enterprise.extensions.xt_project_y.MyPjmbasicValidateObjects
Java Method: ValidateAcctGrp — Execute Extension Validation Class: Before (Before / After)
Space Rules: Not used — Security Type:

New | Clone | Delete | Reset

Verify | Apply | Ok | Cancel

- When extending existing Deltek standard Object Validation, you can create the plug-in class to be executed before or after the standard Deltek Object Validation is fired.
- Enter the full package name of the plug-in class. Enter the method name.
- Select Before or After execution of the standard Deltek Object Validation.
- Implement the class and the method.

Example: Object validation

```
package com.deltek.enterprise.extensions.xt_project_y;
import com.deltek.enterprise.system.applicationinterface.*;
public class MyPjmbasicValidateObjects {
public short ValidateAcctGrp(ResultSetInterface rsI) throws SQLException, DEExcep
tion {
sqlMgr = rsI.getSqlManager(this);
RowSetInterface rowsetI = rsI.getRowSet();
/* In object validation, rsi.getObjectId() method
returns the object Id of the object being validated */
String sAcctGrpObjId = rsI.getObjectId();
String sAcctGrpCd = rowsetI.getStringValue(sAcctGrpObjId);
short nReturn = CP_OK;
String sSql = new StringBuffer()
.append("SELECT ACCT_GRP_CD FROM ACCT_GRP_CD ")
.append("WHERE ACCT_GRP_CD = :sAcctGrpCd ")
.toString();
if (!(sqlMgr.SqlExecuteQuery(sSql))) {
String sMsgId = "CP_ACCTGRP_INVALID";
rsI.addObjectMessage(sAcctGrpObjId, sMsgId ,rsI.ERROR);
nReturn = rsI.ERROR;
}
return nReturn;
}
}
```

Row Validation

Row Validation should be used to validate each row found in a result set (RS).

Timing

In Field mode and Line mode, Row Validation is invoked when at least one field on the row has changed and user moves the focus to another row.

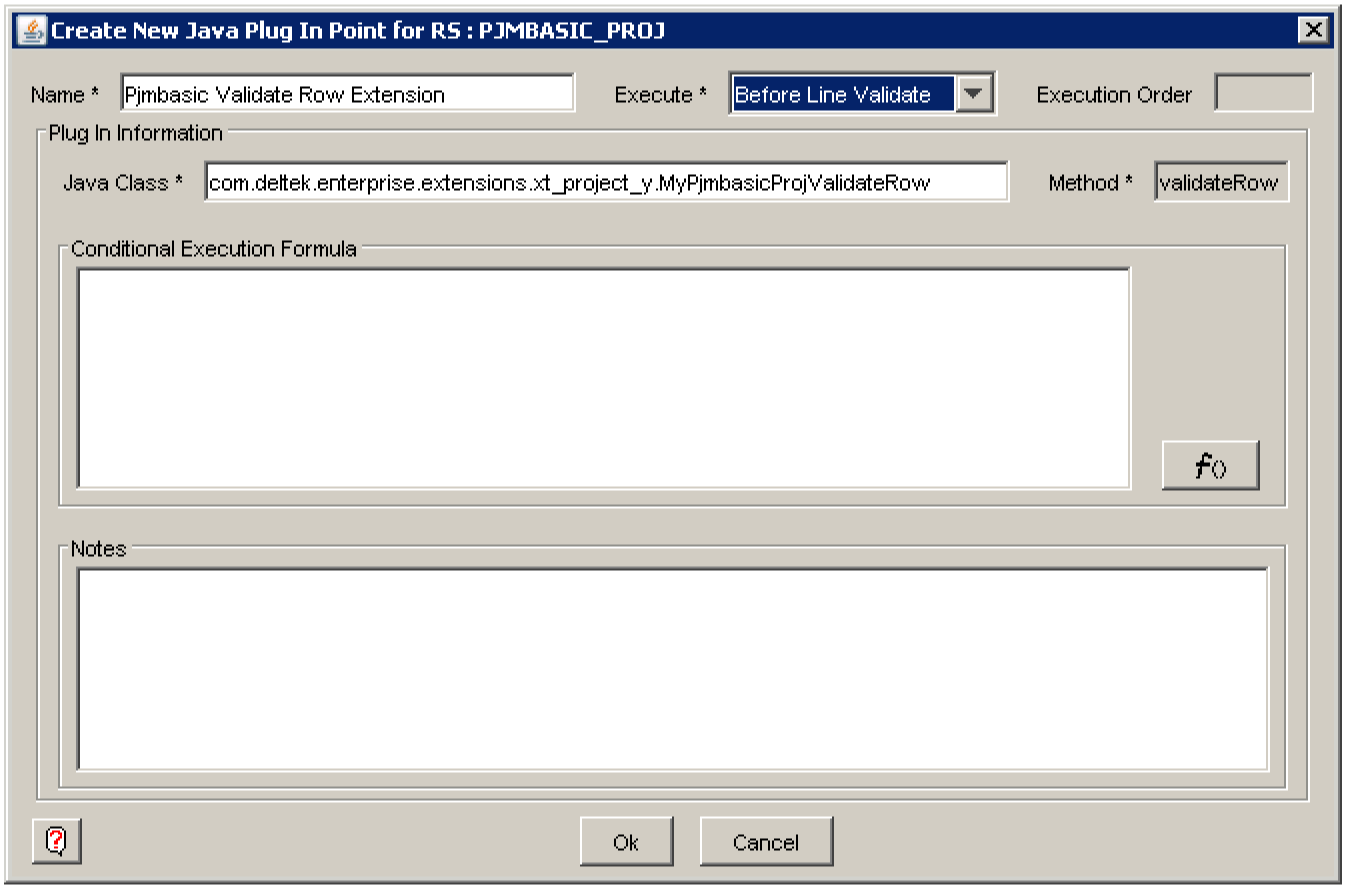
In Application mode or web service mode, Row Validation is delayed and is only called on Save.

Rules

- Row Validation is never called when the row is marked deleted. Therefore, you should never have Row Validation code that checks if the line is marked deleted because the code would never be executed. If you need to check this, do this check in the RS Validation plug-in class.

- Row validation should be written to validate objects' relationship within a line. If the validation logic depends on the value of other objects on other lines (like getting a total, checking for duplicate value, and so on), you should put this logic in RS validation.
- If you have parent and child RS relationship and parent ID or key columns needs to be cascaded/copied down to the child RS on insert of new rows, you need to do this in places other than in the row validation of the child RS.
  The reason is in Field or Record mode, row validation of the child RS may occur and then user decides to change the parent column value (for example, PROJ_ID, VEND_ID, etc,). Since child row is already validated, it will not get validated again when the user changes the value in the parent RS

Extensibility


Create New Java Plug In Point for RS : PJMBASIC_PROJ

Name * Pjmbasic Validate Row Extension

Execute * Before Line Validate

Execution Order

Plug In Information

Java Class * com.deltek.enterprise.extensions.xt_project_y.MyPjmbasicProjValidateRow

Method * validateRow

- When extending existing Deltek standard Row Validation, you can create the plug-in class to be executed before or after the standard Deltek Row Validation is fired. If you need to do both, create two plug-in classes, one for each.
- Enter a descriptive name for the plug-in. Select Before Line Validate or After Line Validate.
- Enter the full package and name of the plug-in class. Note that the method name is always "validateRS".
- The class must implements RSValidation interface with the method validateRS.

Example: Row Validation

```
package com.deltek.enterprise.extensions.xt_project_y;
import com.deltek.enterprise.system.applicationinterface.*;
public class MyPjmbasicProjValidateRow implements RowValidation {
public short validateRow(ResultSetInterface rsI) throws SQLException, DEException
{
RowSetInterface rowSetI = rsI.getRowSet();
String sProjId = rowSetI.getStringValue("PROJ_ID");
String sOrgId = rowSetI.getStringValue("ORG_ID");
String sAcctId = rowSetI.getStringValue("ACCT_ID");
return validatePOA(sProjId,sOrgId,sAcctId,rsI);
}
}
```

RS Validation

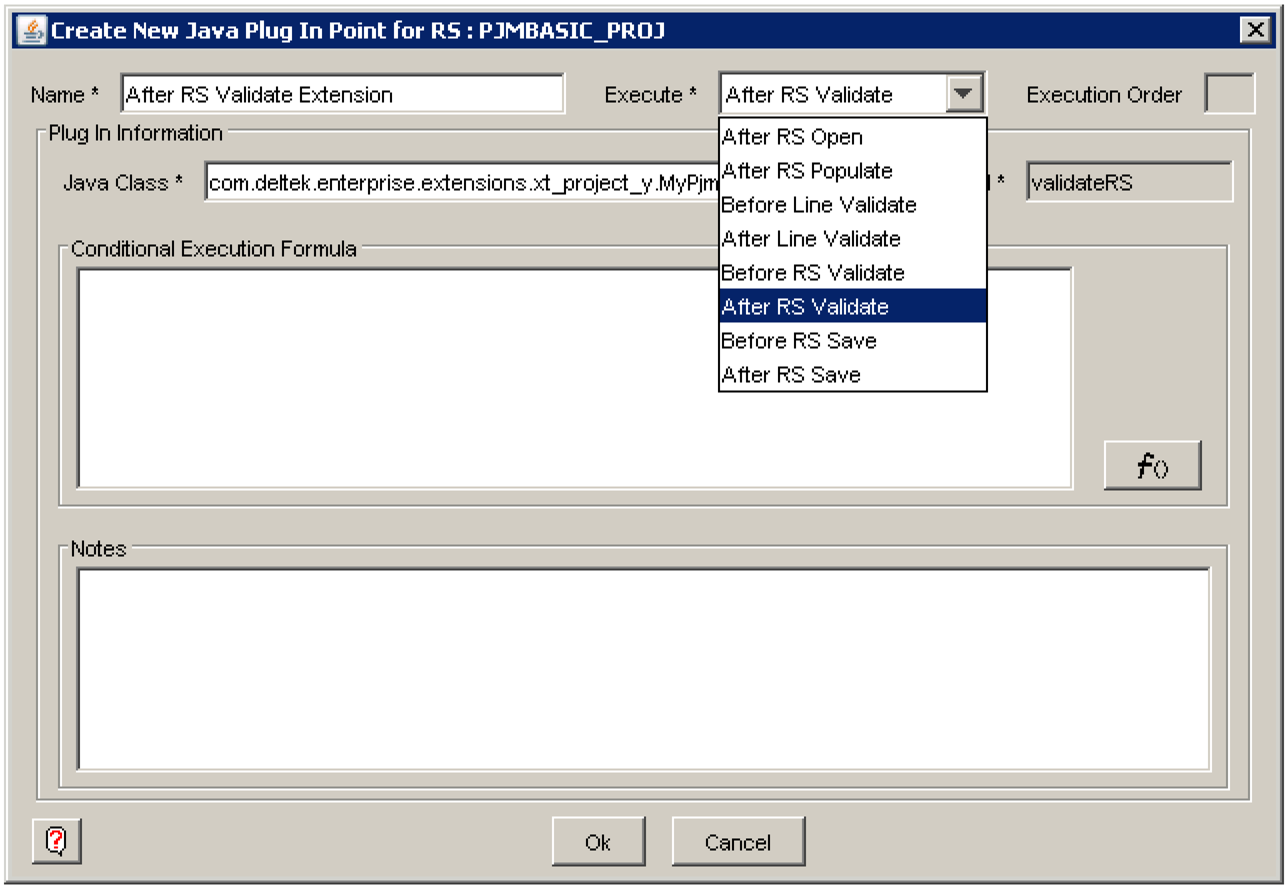
RS Validation should be used to validate across the rows within a RS or across multiple RS.

Timing

In all mode (Field, Row, Application or web service) RS Validation is called only on the Save event.

Rules

- RS Validation is not called if there is no change to any row on that result set.
- RS Validation should be written to validate objects relationship across lines. For example, when totaling amounts or percents across the lines or when checking for unique value of certain column.
- You must always iterate through the RS since the event is fired once for each RS, not each row.



Create New Java Plug In Point for RS : PJMBASIC_PROJ

Name * After RS Validate Extension

Execute * After RS Validate

Execution Order

Plug In Information

Java Class * com.deltek.enterprise.extensions.xt_project_y.MyPjm

validateRS

Conditional Execution Formula

Notes

- When extending existing Deltek standard RS validation, you can create the plug-in class to be executed before or after the standard Deltek RS validation is fired. If you need to do both, create two plug-in classes, one for each.
- Enter a descriptive name for the plug-in. Select Before RS Validate or After RS Validate.
- Enter the full package name of the plug-in class. Note that the method name is always "validateRS".
- The class must implements RSValidation interface with the method validateRS.

Example: RS Validation

```
package com.deltek.enterprise.extensions.xt_project_y;
import com.deltek.enterprise.system.applicationinterface.*;
public class MyPjmbasicValidateRS implements RSValidation {
public short validateRS(ResultSetInterface rsI) throws DEException {
short nReturn = CP_OK;
RowSetInterface rowSetI = rsI.getRowSet();
/* Looping is necessary in RS Validate */
RSIterator rsT = rsI.findInit(rowSetI.ROW_New,rowSetI.ROW_MarkDeleted,true);
int nNextLvlNo = 1;
while (rsT.next() != rowSetI.UNDEFINED_CONTEXT) {
rowSetI = rsI.getRowSet();
int nLvlNo = rowSetI.getInt("PROJ_LVL___LVL_NO");
if (nLvlNo != nNextLvlNo) {
rsI.addLineMessage("PJMBASIC_LVL_NUMBERING", rsI.ERROR);
nReturn = rsI.ERROR;
}
nNextLvlNo += 1;
}
return nReturn;
}
}
```

Connection Mode Summary

In Field mode:

- Object and Row validation occur during data entry when user leaves the field or the line.
- RS validation occurs when you save a record.

In Record mode:

- Row validation occurs during data entry when user leaves the line. Catch up Object validation for all objects occur right before row validations occur.
- RS Validation occurs when you save a record.

In Application mode and web service mode:

- No validation occurs during data entry. Same for web service as the whole data stream is submitted all at once.
- All validations are done when the record is saved.
- Catch up Row Validation for all rows occur right before RS validation occurs. Within Row Validation, catch up Object Validation for all objects occur before row validation occurs.

Frequency Mode	Instant Field Validation	Instant Row Validation
Field Mode	Y	Y
Record Mode	N	Y
Application Mode	N	N

Frequency Mode	Instant Field Validation	Instant Row Validation
Web Service Mode	N	N

Validations Handled by System

There are validations that are automatically handled by the system that you should be aware of before you start coding your validation:

- System will check for nulls in required columns.
- Validates the max length for each data field submitted and that the data matches the field type (for example, string, number, date).
- Validate for entries in a combo box (in case the data is being sent from a non browser client)
- If validated lookups were defined for the result set, then the system does the lookup validations before invoking app-specific validations.
- Other system validation built in to templates (such as X >0 template, and so on)

Before & After Save

During the save transaction, there are three basic events: BeforeSave, Standard Save and AfterSave.

Standard Save is done by the framework on 'saveable' tables represented in the RS (saveable means RS is checked for standard save and PK is present for the table). These are set in the Designer for the result set. There is no plug-in for Standard Save. It is done entirely by the framework.

If you need to update other tables not represented in the RS, you can utilize the "beforeRSSave" or the "afterRSSave" event. For example, update inventory after a purchase order is saved on the Manage Purchase Order application.

If there is more than one result set in an application, beforeRSSave are issued first to the top result set and then down until the last result set at the bottom of the tree.

After BeforeRSSave for all result set are completed, Standard Save of the result set(s) will take place. Again, if there are more than one result set in an application/tree, Standard Save are done first to the top result set and then down until the last result set at the bottom of the tree. Within the standard Save for a single result set, Delete happens first, then Update and then Insert.

After standard Save for all result set are completed, AfterRSSave will take place. Again, if there are more than one result set in an application, afterRSSave are done first to the top result set and then down until the last result set at the bottom of the tree.

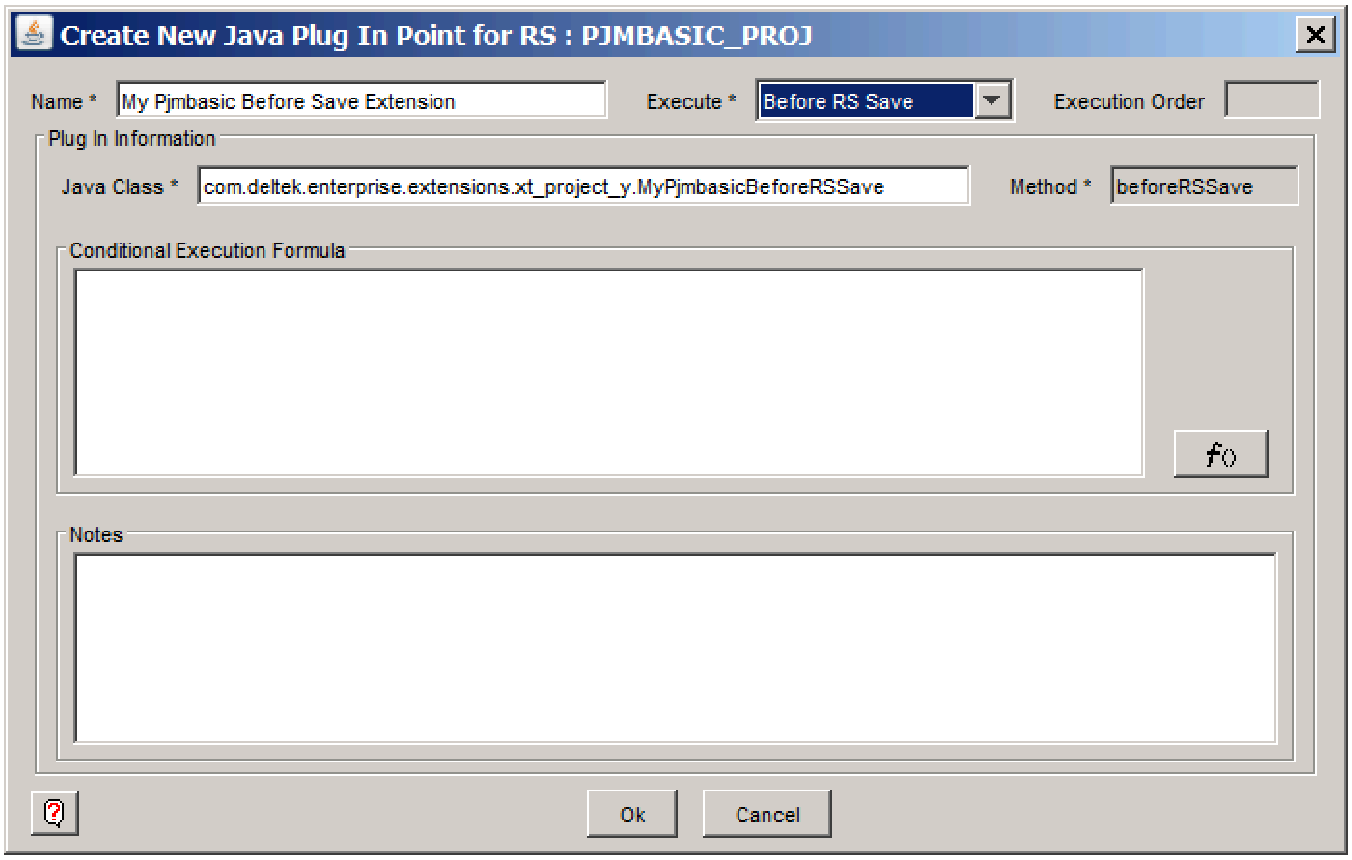
BeforeRSSave

Timing

- BeforeRSSave is invoked only during a save transaction. It is never invoked during data entry regardless of connection mode.
- After all the validations are done, the system will start the database transaction. Then it calls the BeforeRSSave plug-in class.

Rules

- BeforeSave is used to save data to tables other than the main table (if these tables are needed before the main table can be saved).
- There should be no commit in this method since transaction has begun.
- BeforeRSSave can also be used to assign parent keys/ID to the children RS if the ID are not assigned in validation for fear of having gaps in ID when validation fails. If BeforeRSSave fails for some reason, all update in it will be rolled back.
- Although not recommended, you can perform validation in this plug-in and if error severity message is added to the message container (via method addObjectMessage or addRowMessage, and so on) the framework will rollback all updates done so far in the save event.


Create New Java Plug In Point for RS : PJMBASIC_PROJ

Name * My Pjmbasic Before Save Extension

Execute * Before RS Save

Execution Order

Plug In Information

Java Class * com.deltek.enterprise.extensions.xt_project_y.MyPjmbasicBeforeRSSave

Method * beforeRSSave

Conditional Execution Formula

Notes

- Enter a descriptive name for the plug-in. Select Before RS Save.
- Enter the full package and class name of the plug-in class. Note that the method name is always "beforeRSSave".
- The class must implements BeforeRSSave interface with the method beforeRSSave.
- Since this method is fired once for each rowset, looping is necessary to loop through the rowset.

Example: BeforeRSSave

```
package com.deltek.enterprise.extensions.xt_project_y;
import com.deltek.enterprise.system.applicationinterface.*;
public class MyPjmbasicBeforeRSSave implements BeforeRSSave{
public String sProjId = "";
/* Never commit in a beforeRSSave plug-in */
public short beforeRSSave(ResultSetInterface rsI) throws DEException {
SqlManager sqlMgr = rsI.getSqlManager(this);
RowSetInterface roI = rsI.getRowSet();
/* Looping is necessary */

RSIterator iT = rsI.findInit(roI.ROW_New, roI.ROW_MarkDeleted,true);
while (iT.next() != roI.UNDEFINED_CONTEXT) {

  sProjId = roI.getStringValue("PROJ___PROJ_ID");

if (roI.isRowNew()) {

..call some private method......
    }
  }
return CP_OK;
}
}
```

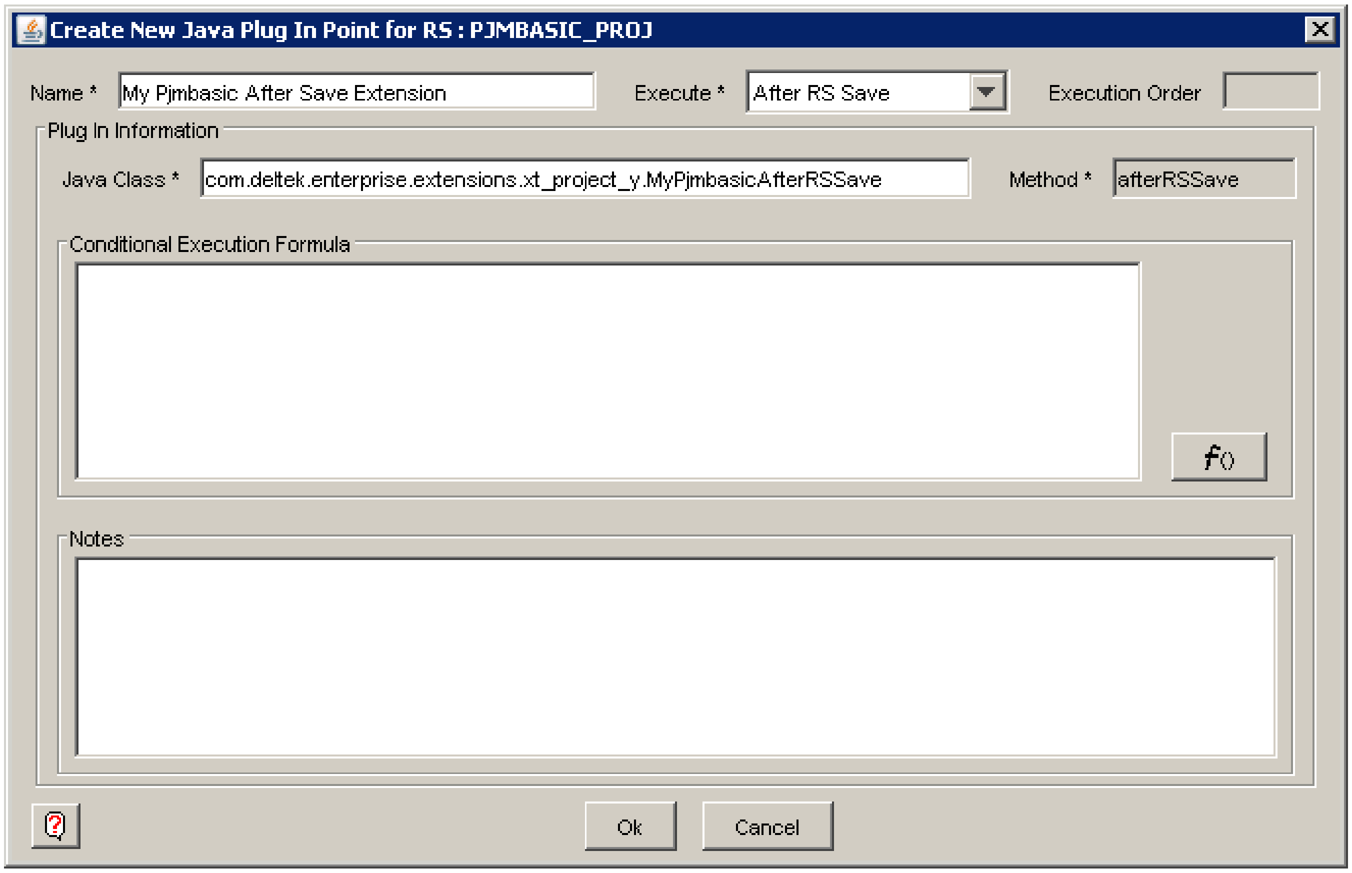
AfterRSSave

Timing

- AfterRSSave is invoked only during a save transaction. It is never invoked during data entry regardless of connection mode.
- After the BeforeRSSave and standard Save are executed, the system calls the AfterRSSave plug-in class.

Rules

- AfterRSSave is used to save data to tables other than the main table (if these tables are needed after the main table is saved). For example: Saving a new PROJ requires saving a new PROJ_EDIT.
- There should be no commit in this method since it is still within the transaction.
- Although not recommended, you can perform validation in this plug-in and if error severity message is added to the message container (via method addObjectMessage or addRowMessage, and so on) the framework will rollback all updates done so far in the save event.
- Tip: AfterRSSave can be used to select data updated in BeforeRSSave and standard Save if the it is more efficient with SQL than with java code. This is possible since the connection is within the same transaction. For example, select amount from header table and compare with total from all its children line.


Create New Java Plug In Point for RS : PJMBASIC_PROJ
Plug In Information
Conditional Execution Formula
Notes
f()
Ok
Cancel

Name * My Pjmbasic After Save Extension
Execute * After RS Save
Execution Order
Java Class * com.deltek.enterprise.extensions.xt_project_y.MyPjmbasicAfterRSSave
Method * afterRSSave

- Enter a descriptive name for the plug-in. Select After RS Save.
- Enter the full package and class name of the plug-in class. Note that the method name is always "afterRSSave".
- The class must implements AfterRSSave interface with the method afterRSSave.
- Since this method is fired once for each rowset, looping is necessary to loop through the rowset.

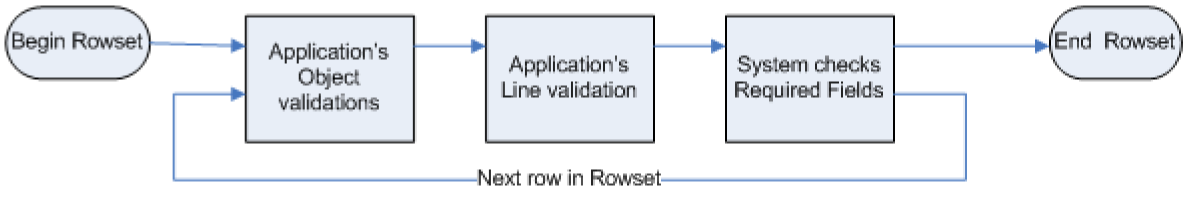
Example: AfterRSSave

```
package com.deltek.enterprise.extensions.xt_project_y;
import com.deltek.enterprise.system.applicationinterface.*;
public class MyPjmbasicAfterSave implements AfterRSSave {
public String sProjId = "";
public String sActiveFl = "";
private String sOrigActiveFl = "";
public short afterRSSave(ResultSetInterface rsI) throws Exception {
/* Never commit in a afterRSSave plug-in */
SqlManager sqlMgr = rsI.getSqlManager(this);
RowSetInterface roI = rsI.getRowSet();
sProjId = roI.getStringValue("PROJ___PROJ_ID");
sActiveFl = roI.getStringValue("PROJ___ACTIVE_FL");

/* Looping is necessary */

RSIterator iT = rsI.findInit(roI.ROW_New, roI.ROW_MarkDeleted,true);
while (iT.next() != roI.UNDEFINED_CONTEXT) {
if (roI.isRowModified()) {
sOrigActiveFl = roI.getStringValue("ORIG_ACTIVE_FL");
if (!(sActiveFl.equals(sOrigActiveFl))) {
String sSql = "UPDATE PROJ_ORG_ACCT SET ACTIVE_FL = :sActiveFl " +
"WHERE PROJ_ID = :sProjId";

    sqlMgr.SqlPrepareAndExecute(sSql);
      }

    }
  }
sqlMgr.close();
return CP_OK;
}
}
```

Order on Save

Since there are many validation and save plug-in events, it is worthwhile to review the sequence.

Validation sequence (intra result set)

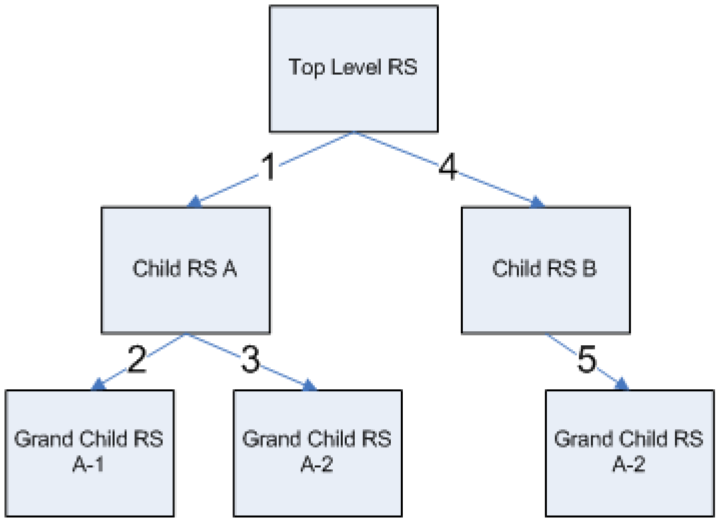

Begin Rowset
Application's Object validations
Application's Line validation
System checks Required Fields
End Rowset
Next row in Rowset

- For each line in a rowset, object validation is done first (in the order the objects are displayed in the DESC tab, unless overridden by the validation sequence).
- Row validation is done next.
- System checks for all required fields on that line
- Go to next row in rowset.

- When a rowset is done, system performs RS validation
- Then move to the next rowset until all the rowset for the RS is finished
- Example: Top level Result set has two rows: Row 1 and Row 2. Object Validation will be done for row 1, then Line Validation for row 1, and then Required fields for row 1. Moving to Row 2: Object will be done for row 2, then Line Validation for row 2, then Required fields for row 2. When both rows are validated, then RS validation will be called.

Validation sequence (inter result sets)

- If there is more than one result set in an application, the system traverses down the tree and completes the intra RS validation (described above) one result set at a time.



- The order of the children selected (when they are at the same level) is not determinate. That is, there is no significance to choose Child RS A or Child RS B first.

Save sequence

- Once validation for all result sets is completed, the save transaction starts.
- Extensibility Before RS Save plug-in is executed. If there are more than one result set, the inter result set set sequence is the same as described above.
- Then standard Deltek Before RS Save is executed (for all result sets).
- Then standard RS Save is executed. Delete is done first, then Update, then Insert.
- Then After RS Save is executed
- Then Extensibility After RS Save is executed.
- Save transaction completes.

Actions

An Action is a set of logic that is executed on the request of the user. It is in addition to the standard event such as RS Populate, Validation or Save.

Usually action is triggered via application specific buttons on the application screen or via the drop down from the Gear icon on the main menu.

ActionInterface

Unlike other events where the plug-in class receives the handle to the ResultSetInterface, all action plug-in class receives the handle to the ActionInterface. From ActionInterface, you can obtain ResultSetInterface and from there do your normal data retrieval logic. ActionInterface contains additional methods to handle long running process such as displaying progress meter dialog, controlling database transaction, and so on.

Here are some commonly used methods provided by the interface (see javadoc for more explanations)

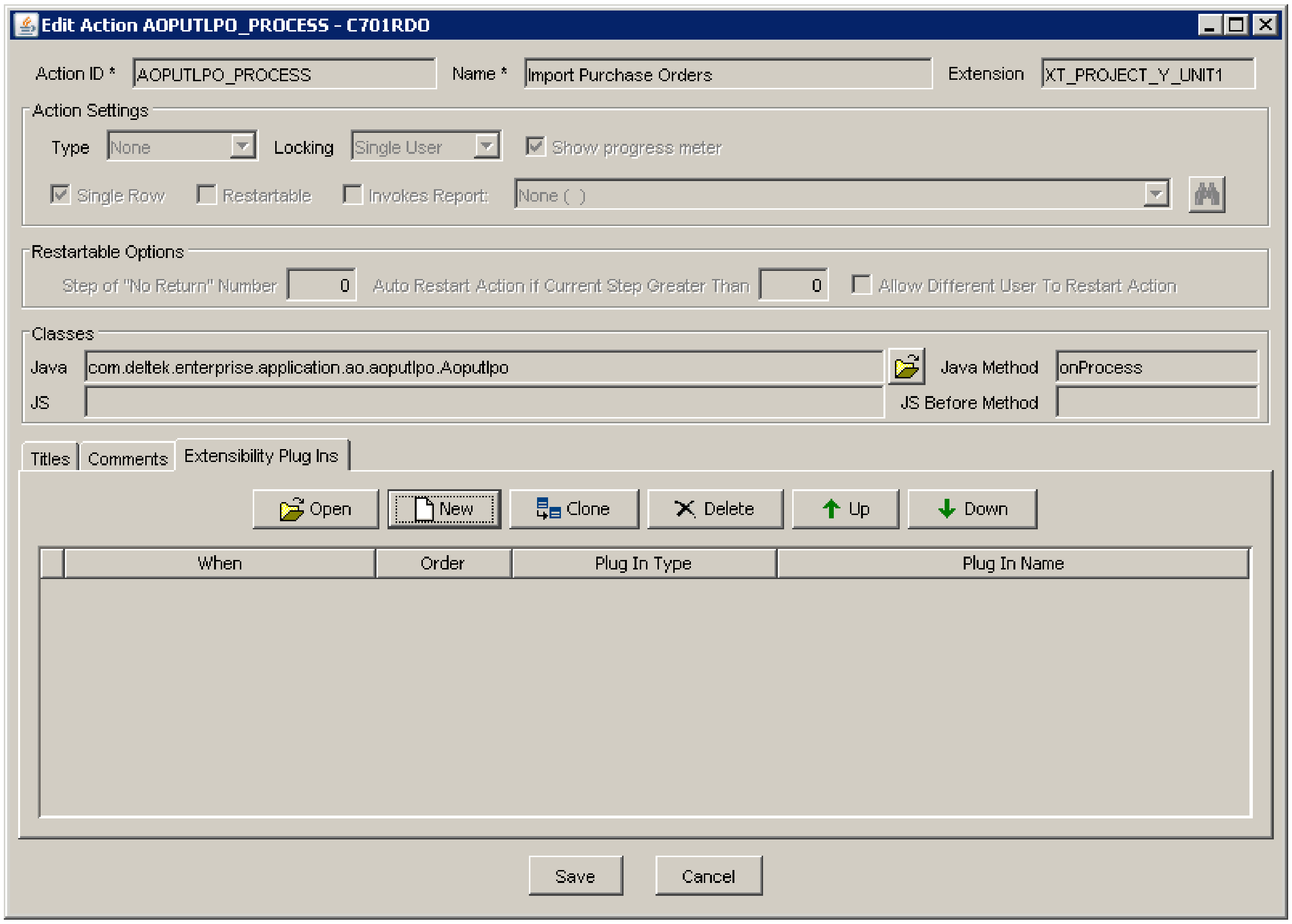
- getResultSet(): Return the ResultSetInterface of the screen the action is assigned to in the Designer. For a process, this is usually the parameter screen.
- getApplication(): Return the AppInterface for the application this action was invoked from.
- getSqlManager(): Return an instance of SqlManager
- getFileManager(): Return an instance of FileManager to access disk file

With Extensibility, you can extend an existing Deltek action or add a brand new action.

Web Services

If you are interested in invoking Costpoint Web Services from Extensibility plug-ins, refer to "Appendix C: Example of Invoking Generated Web Service from inside Costpoint Extensibility Code" in the *Deltek Costpoint Integration Overview* guide.

Extending a Standard Deltek Action


Edit Action AOPUTLPO_PROCESS - C701RDO

Action ID * AOPUTLPO_PROCESS Name * Import Purchase Orders Extension XT_PROJECT_Y_UNIT1

Action Settings

Type None Locking Single User ✓ Show progress meter

✓ Single Row ☐ Restartable ☐ Invokes Report: None ( )

Restartable Options

Step of "No Return" Number 0 Auto Restart Action if Current Step Greater Than 0 ☐ Allow Different User To Restart Action

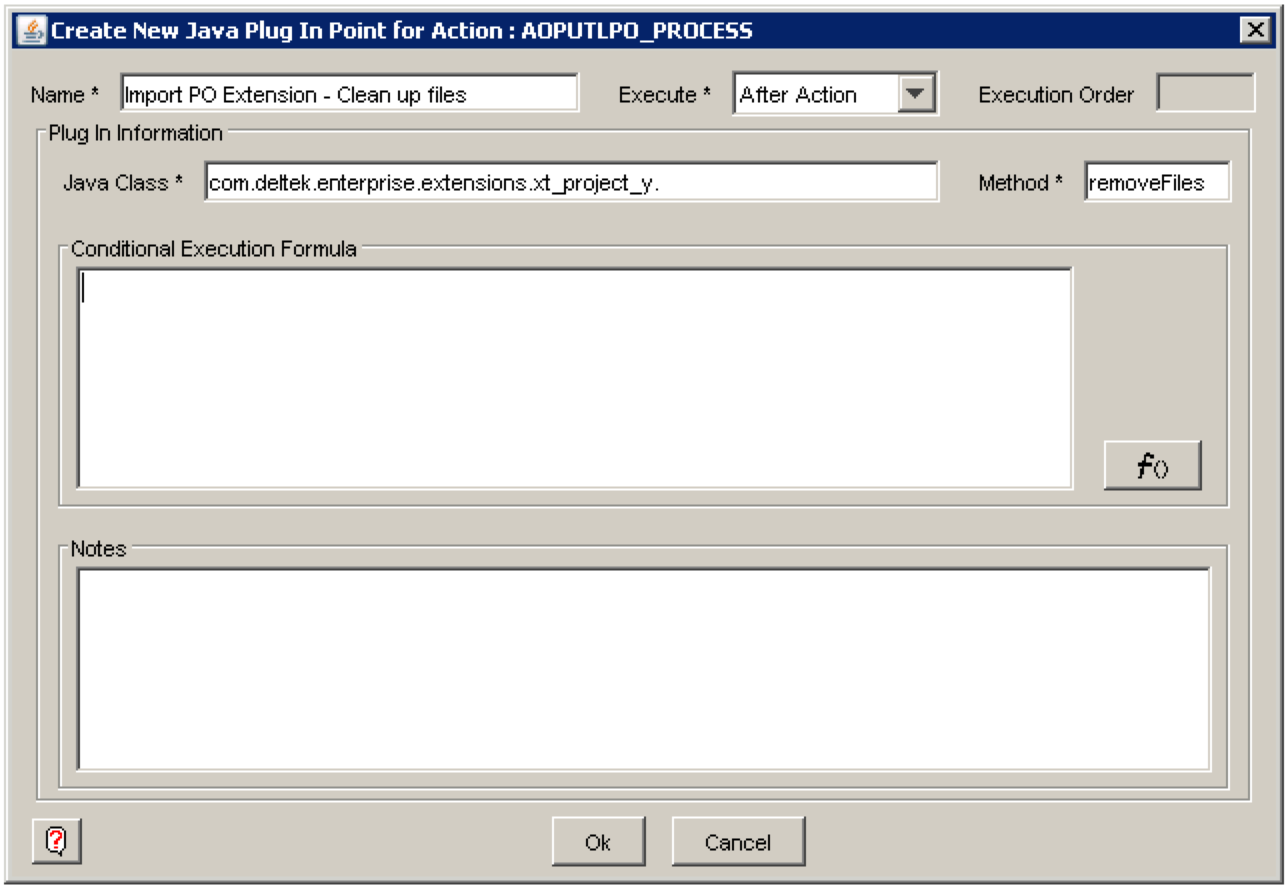
Classes

Java com.deltek.enterprise.application.ao.aoputlpo.Aoputlpo Java Method onProcess

JS JS Before Method

	When	Order	Plug In Type	Plug In Name

- In the Designer, select **Project » Unit » Extend Action**, and select action to extend.
- Select Extensibility Plug Ins tab and create the Java extension for the action.


Create New Java Plug In Point for Action : AOPUTLPO_PROCESS
Name * Import PO Extension - Clean up files
Execute * After Action
Execution Order
Plug In Information
Java Class * com.deltek.enterprise.extensions.xt_project_y.
Method * removeFiles
Conditional Execution Formula
Notes
Ok
Cancel

- Enter a descriptive name for the plug-in. Select Before Action or After Action to be executed before or after the standard Deltek action logic.

> **Note:** If you add a Message with a Severity ERROR or FATAL inside your Before Action plug-in, all the subsequent Before Action plug-ins and main Action will be canceled. After Action plug-ins will be executed, regardless of the severity of the messages in the container. You can use the method isProcessCanceled in ActionInterface to check if the main process was canceled or not.

- Enter the full package and class name of the plug-in class. Note that the method name can be any name since we are not implementing any required interface like with RowValidation , RSValidation or Before or AfterRSSave. The method is public and return a boolean.

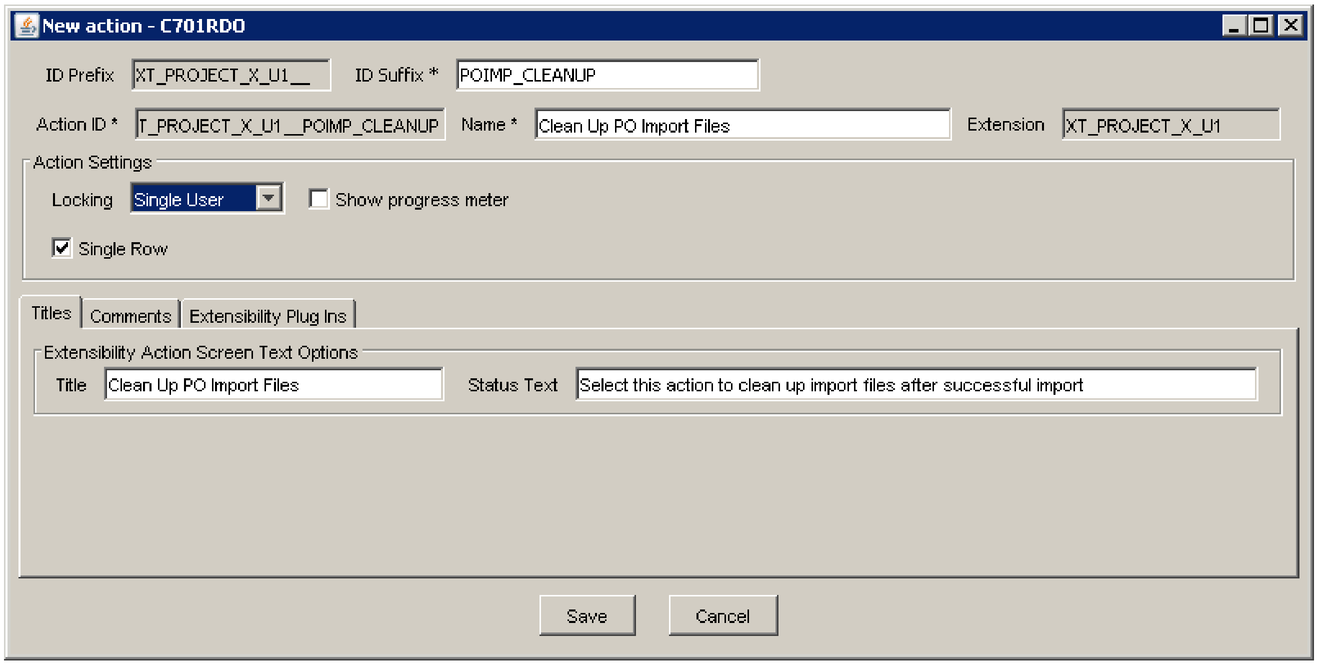
Example: Using FileHandlerInterface

```
package com.deltek.enterprise.extensions.xt_project_y;
import com.deltek.enterprise.system.applicationinterface.*;
public class ImportPO {
public boolean removeFiles (ActionInterface actI) throws DEException {

  ResultSetInterface rsI = actI.getResultSet();
  RowSetInterface roI = rsI.getRowSet();
  String fileName = roI.getStringValue("INPUT_FILE_NAME");
  FileHandlerInterface fileI = actI.getFileManager();
  if (!fileI.deleteFile(fileName)) {
  rsI.addObjectMessage("INPUT_FILE_NAME", "XT_PROJECT_Y_UNIT1__DEL_FILE_ERR", rs
I.ERROR);
  return false;
        }
  return true;
  }
}
```

Adding a new extensibility action

The steps to add new action to an existing application are very much the same as extending an existing Deltek action.

- In the Designer, select **Project » Unit » New » Action**.
- Define the action.


New action - C701RDO

ID Prefix: XT_PROJECT_X_U1__
ID Suffix *: POIMP_CLEANUP
Action ID *: T_PROJECT_X_U1__POIMP_CLEANUP
Name *: Clean Up PO Import Files
Extension: XT_PROJECT_X_U1

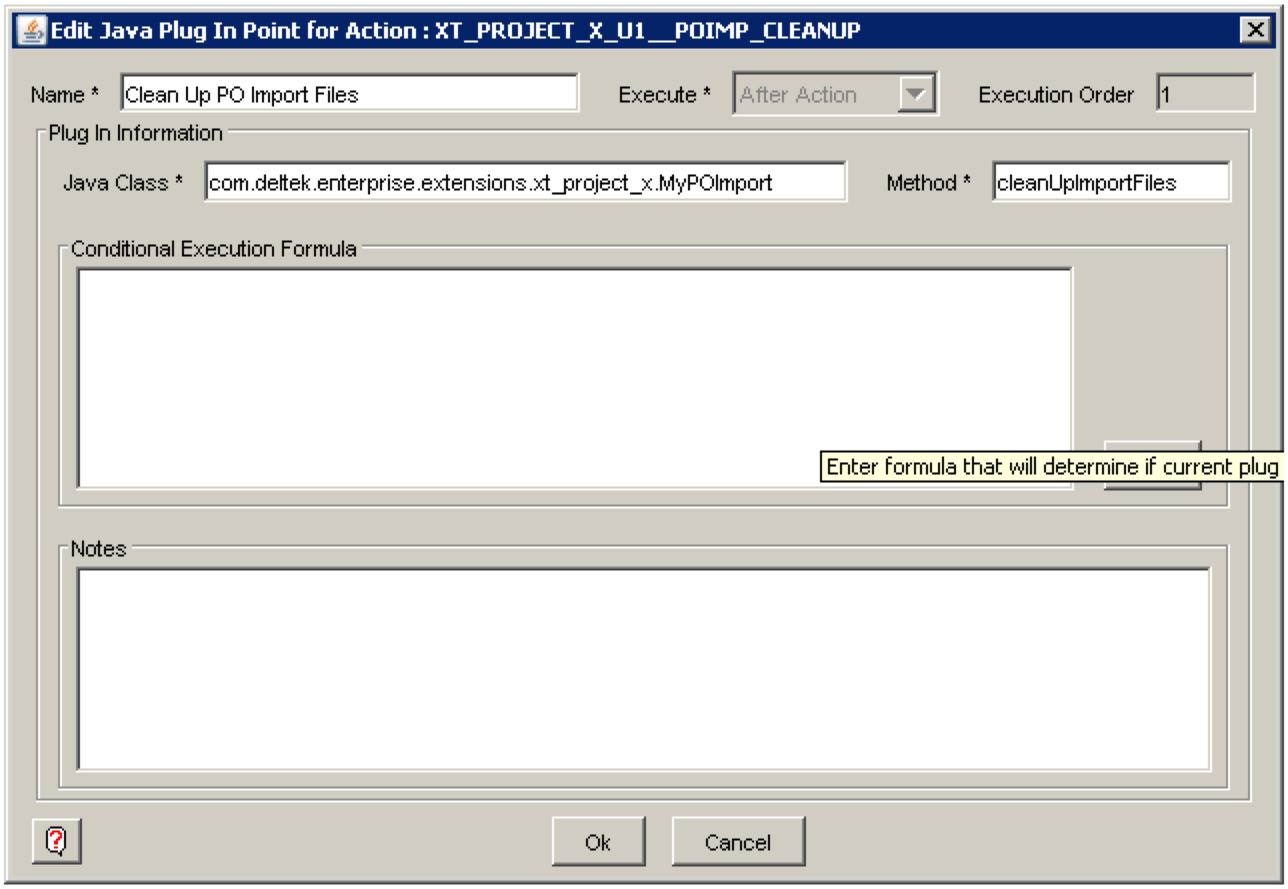
Action Settings
Locking: Single User
[ ] Show progress meter
[✓] Single Row

Titles | Comments | Extensibility Plug Ins

Extensibility Action Screen Text Options
Title: Clean Up PO Import Files
Status Text: Select this action to clean up import files after successful import

Attention: See Extensibility Designer User Guide for explanation of fields on this screen.

- Add Java plug-in for the action and specify the class name and method.


Edit Java Plug In Point for Action : XT_PROJECT_X_U1__POIMP_CLEANUP

Name * Clean Up PO Import Files

Execute * After Action

Execution Order 1

Plug In Information

Java Class * com.deltek.enterprise.extensions.xt_project_x.MyPOImport

Method * cleanUpImportFiles

Conditional Execution Formula

Enter formula that will determine if current plug

Notes

- Notice that there is no selection for Execute mode. There is no before or after selection. Your java class plug-in executes by itself without any standard Deltek action class since the action is created by you.
- Once you have the new action defined, the java class is implemented normally like you have for extended action.

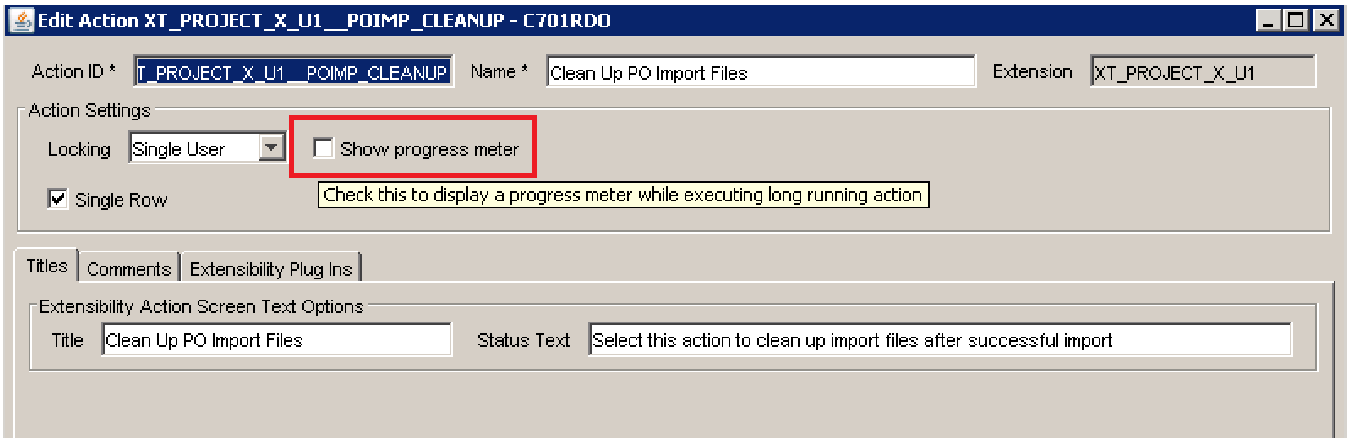
Example: Using FileHandlerInterface

```
package com.deltek.enterprise.extensions.xt_project_y;
import com.deltek.enterprise.system.applicationinterface.*;
public class MyPOImport {
public boolean cleanUpImportFiles (ActionInterface actI) throws DEException {

  ResultSetInterface rsI = actI.getResultSet();
  RowSetInterface roI = rsI.getRowSet();
  String fileName = roI.getStringValue("INPUT_FILE_NAME");
  FileHandlerInterface fileI = actI.getFileManager();
  if (!fileI.deleteFile(fileName)) {
  rsI.addObjectMessage("INPUT_FILE_NAME", "XT_PROJECT_Y_UNIT1__DEL_FILE_ERR", rs
I.ERROR);
```

```
    return false;
       }
    return true;
    }
  }
```

Action Progress Meter

To show the action progress meter when an action is executed, the "Show Progress Meter" must be checked.


Edit Action XT_PROJECT_X_U1__POIMP_CLEANUP - C701RDO

Action ID * T_PROJECT_X_U1__POIMP_CLEANUP   Name * Clean Up PO Import Files   Extension XT_PROJECT_X_U1

Action Settings

Locking Single User   ☐ Show progress meter

Check this to display a progress meter while executing long running action

☑ Single Row

Titles | Comments | Extensibility Plug Ins

Extensibility Action Screen Text Options

Title Clean Up PO Import Files   Status Text Select this action to clean up import files after successful import

Text for Progress Meter

To enable internationalization of text shown on the progress meter, text must be entered in the Resource Message in the Designer. Then in application java code, use the appropriate method from ActionInterface to set the text by passing the Message ID (see below).

Methods for Progress Meter

Common methods for manipulating progress dialog are available from ActionInterface:

- void **setDlgMeterLimit**(int meterLimit): Sets the status meter limit
- void **SetDlgMeterValue** (int meterValue): Sets the status meter value
- void **setDlgCountText**(java.lang.String sText): Sets the action status count text property
- void **setDlgCount**(int nCount): Sets the action status count
- void **setDlgMeterTextLine**(int lineNo, java.lang.String meterTextID): Sets the status text line to the text ID in Resource Messages.
  LineNo = ActionInterface. DLG_TOP_LINE or ActionInterface.DLG_ BOTTOM_LINE
  Use DLG_TOP_LINE for major steps and DLG_ BOTTOM_LINE for minor steps.

```
actionI.setDlgMeterTextLine (ActionInterface.DLG_TOP_LINE, "XT_PROJECT_X_UNIT
1_TOP_TEXT");
actionI.setDlgMeterTextLine (ActionInterface.DLG_BOTTOM_LINE, "XT_PROJECT_X_UNI
T1_BOT_TEXT");
```

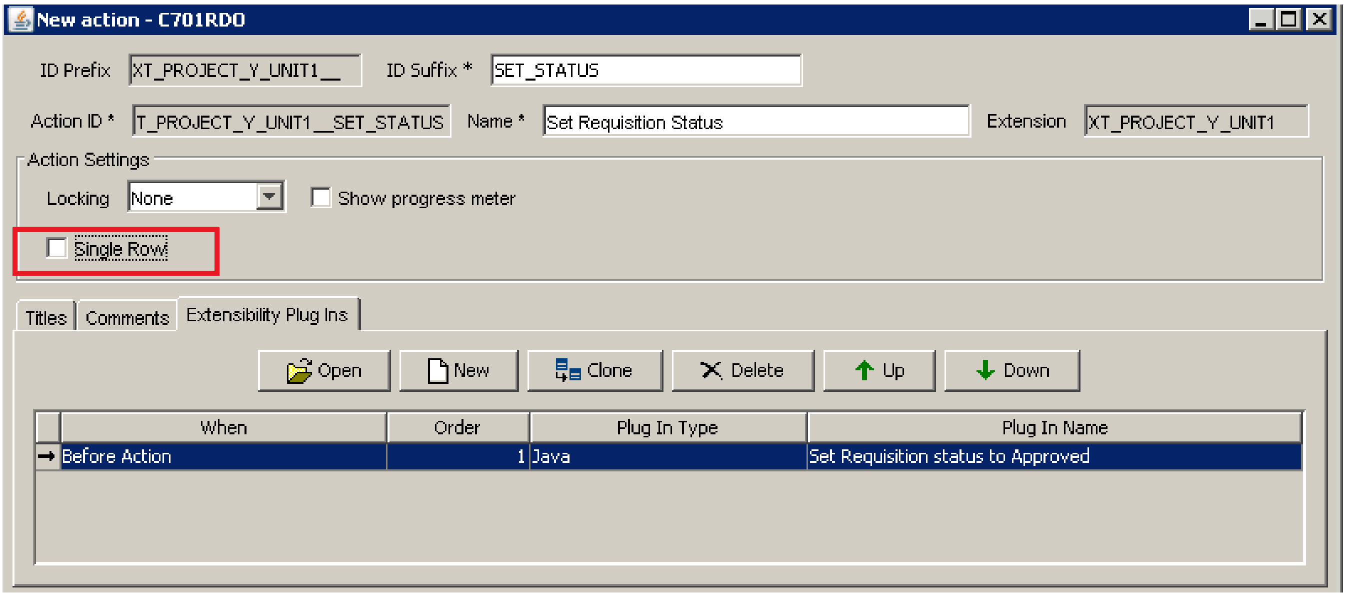
- boolean **checkUserCancel**(): Checks if the action is being cancelled by user. Returns TRUE if the action was cancelled, FALSE otherwise. Call this function as often as necessary to check if user has tried to cancel.
- public void **setDlgCancelText**(int optionNo): Set the confirm message to display when user cancels the action. Set optionNo to either CANCEL_TXT_HARD to display a stronger warning that some table have been updated or to CANCEL_TXT_SOFT for a normal confirmation text.
- void **addMessage**(String msgId,short msgType): Add messages to be displayed at the end of the action (or when action fails). These messages will be displayed at the bottom similar to validation messages. If any of the messages has severity of ERROR or higher, the system treats the process as un-successful.

Example: Using progress meter

```
public class MyAction {
private SqlManager sqm;
private ActionInterface actI;
public short **processWithDialog** (ActionInterface actionI) throws DEException,S
QLException {
actI = actionI;
actI.setDlgMeterLimit(100);
actI.setDlgMeterTextLine (actI.DLG_TOP_LINE, "CP_YDLG_PREPROC_DATA");
actI.setDlgMeterTextLine (actI.DLG_BOTTOM_LINE, CP_YDLG_CREATE_WRK_TBL);
sqm = actI.getSqlManager(this);
if (!doStep1())
return false;
if (!doStep2())
return false;
... more steps ...
actI.setDlgMeterValue(100);
return true;
}
private boolean doStep1() throws DEException, SQLException {

  actI.setDlgMeterValue(10);
  if (funcCheckUserCancel())
  return false;
  ... do Step 1 ...
  sqm.SqlCommit();
  return true;
}
/* Check for user cancelling */
private boolean funcCheckUserCancel() throws DEException{

  if (actI.checkUserCancel()) {
  actI.setDlgMeterTextLine (actI.DLG_TOP_LINE, "CP_YDLG_PROC_CANCEL");

      actI.setDlgMeterTextLine(actI.DLG_BOTTOM_LINE,"");
      return true;
  }
  return false;
}
}
```

Maintenance Action

Action can be used on a maintenance / transaction application to aid in faster data entry. These generally do not process or update data in the database. They are mainly used for defaulting or setting values to the data user has on the screen. These are considered maintenance action. They are not long running action.


New action - C701RDO

ID Prefix: XT_PROJECT_Y_UNIT1__
ID Suffix *: SET_STATUS
Action ID *: T_PROJECT_Y_UNIT1__SET_STATUS
Name *: Set Requisition Status
Extension: XT_PROJECT_Y_UNIT1

Action Settings
Locking: None
[ ] Show progress meter
[ ] Single Row

When	Order	Plug In Type	Plug In Name
Before Action	1	Java	Set Requisition status to Approved

Single Row is usually checked for long running action and generally is not checked for a maintenance action. When it is not checked, the code needs to loop through the rows in the result set.

Example: Loop through rows selected by user and change Requisition status to Approved

```
package com.deltek.enterprise.xt_project_y.req;
import com.deltek.enterprise.system.applicationinterface.*;
public class RqLineAction {
public boolean setStatus (ActionInterface acI) throws DEException {
/* get ResultsetInterface from ActionInterface */
ResultSetInterface rsI = acI.getResultSet();
RowSetInterface roI = null;

/* Create iterator that looks for row selected and exclude row mark deleted */

RSIterator rst = rsI.findInit(roI.ROW_Selected,roI. ROW_MarkDeleted, true);
roI = rsI.getRowSet();

/* Loop through rows and set values */
while (rst.next()!= roI.UNDEFINED_CONTEXT) {
roI.setStringValue("A","S_RQ_STATUS_CD");
  }
return true;
  }
}
```

In Costpoint 2026.1, we introduced a new type of extensibility plugins which allows you to extend the Ask Dela assistant functionality. The overall paradigm closely follows the industry standard MCP approach, with Ask Dela acting as the MCP Client and the extensibility plugin acting as the MCP tool/server registered with Ask Dela. Given that in this scenario, the MCP client and MCP server both operate within a unified Costpoint platform, we can streamline or simplify tool development and registration for extensibility developers, skipping or simplifying steps which would typically be required in the case of a 3rd party MCP development process. Also, given that Dela tools are created with the goal to use or expose Costpoint capabilities, simple APIs are offered to an "out of the box" extensibility developer to do typical tasks, e.g. check if a user has permissions to a specific Costpoint application or programmatically perform specific tasks in a Costpoint screen.

While we may colloquially refer to this new type of extensions as "AI functions" or "AI agents," the documentation below explains different types of such functions, what they can do, and provides a few coding examples. This is a rapidly evolving area, and as we build or mature additional AI capabilities used by Costpoint developers, we will be gradually exposing the same capabilities to Extensibility developers and updating this section.

How AI Functions Work

When you interact with Ask Dela, the system:

1. Analyzes your question to identify relevant functions

2. Provides AI with a curated list of available functions based on trigger words and context
3. Allows AI to decide whether to call functions and with what parameters
4. Executes functions and return results to AI
5. Allows AI to generate a natural language response based on function outputs

Package Structures for AI agents

Using the structure enforced for the extensions package (see Package Structure in Chapter 1 for more information),

com.deltek.enterprise.extensions.<extensibility project id>, we recommend you add the aiagents folder. This will hold all AI agents Java classes.

Example:

- com.deltek.enterprise.extensions.xt_project_y.aiagents (to be shared by all code under this project)
- com.deltek.enterprise.extensions.xt_project_y.unit1.aiagents (to be shared by all code under unit1)
- com.deltek.enterprise.extensions.xt_project_y.unit1.symusr.aiagents (specific code when creating AI agents classes specific to the SYMUSR application under unit 1)

Create Your AI Function Class

When creating AI functions, developers should extend the com.deltek.enterprise.system.applicationinterface.ai.AIFunction object.

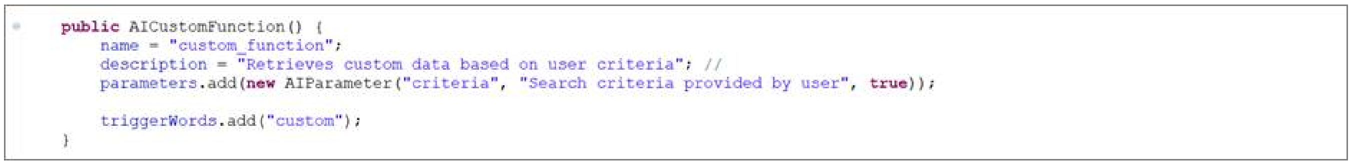
AIFunction is the abstract base class that all AI plugins must extend in the Costpoint AI Framework. It provides the foundational structure for creating AI-powered functions that can be invoked by the AI assistant to answer user questions or perform actions. This class defines the contract for AI function behavior, parameter handling, execution lifecycle, and integration with the framework.

Key Properties

Property	Purpose	Required
Name	Unique function identifier for AI	Yes
Description	Detailed description for AI model	Yes

Property	Purpose	Required
Parameters	List of function parameters	No
triggerWords	Keywords that activate this function	No

Each AI function must have a constructor that declares **name**, **description** and, potentially, one or several **parameters** that AI can use to pass additional information. You should also list **triggerWords** that instruct framework when to use this function.



```
public AICustomFunction() {
    name = "custom_function";
    description = "Retrieves custom data based on user criteria"; //
    parameters.add(new AIParameter("criteria", "Search criteria provided by user", true));

    triggerWords.add("custom");
}
```

- name (String)
  A unique identifier for the function.
  - Used by the AI to reference the function.
  - Must be short, no spaces.

  Example:
  `name = \"create_expense_report\";`
- description (String)
  A natural-language explanation of what the function does.
  - This is part of the "function interface" we present to AI, to help AI know *when* to call this function.
  - It should be clear and specific.
  - When registering the AI function in the Extensibility designer, user is required to provide value in the "Agent description" field. For ease of maintenance and readability of code it is recommended to keep description in-sync between java and the Designer.
  - During runtime, framework initially takes description from ai function java class, then we override it with description from the Designer. In some rare cases, we override it with runtime description from ai function java class when **adjustDefinition()** method is implemented in the ai function (more details on the later part of the document).

  Example

  ```
  description = \"create new expense report with given description. User may abbr
  eviate expense report as ER.\";

  description = \"Retrieves employee information including name, department, and
  contact details based on employee ID or name.\";
  ```
- Parameters (ArrayList<AIParameter>)
  A schema that describes the input the function expects.
  Each **parameter** is represented by:
  AIParameter (String name, String description, boolean required) object with name, description properties and

indicator whether parameter is required. If parameter is required, AI should always pass value into this parameter when it calls function, otherwise AI assumes that parameter is optional and decides whether to pass it based on user question. See AIParameter for more details.
Example

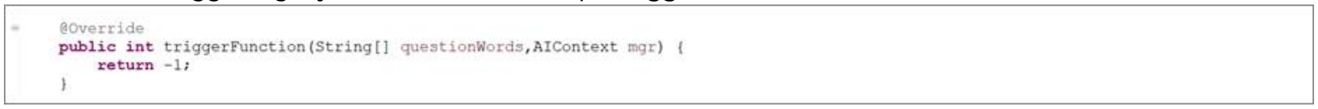
```
parameters.add(new AIParameter(\"expense_report\", \"Expense report to which us
er wants to get expenses . Pass \'current\' when user does not mention specific
report or mentions it by \'current\' or \'latest\'.\",true));
```

- triggerWords
  A list of words that trigger the framework when to use this function.
  - These are keywords that indicate whether the function is relevant to a user's query. Framework scans user input for these words to determine which functions to show AI.
  - You can include synonyms, abbreviations, and related terms. But don't use too generic words (like articles, common words, etc.) as it will trigger unnecessary execution of your plug-ins.
  - You need to be careful when selecting the keywords. If they are widely used within the systems and are used in other AI agents, chances are AI might use other functions rather than yours. However, do not use highly generic words as it will make AI unnecessarily trigger your plug-in to answer every simple question.
  - triggerWords are not strictly required. In more complicated use-cases where triggerWords are not enough, you can potentially implement the **triggerFunction()** method along with trigger words (or even instead of trigger words, see additional details below).

  Example

```
triggerWords.add(\"expenses\");
triggerWords.add(\"expense\");
```

- triggerFunction (String[] questionWords, AIContext mgr)
  An advanced trigger logic you can use when simple **triggerWords** is insufficient.



```
@Override
public int triggerFunction(String[] questionWords,AIContext mgr) {
    return -1;
}
```

This method allows you to analyze the whole user question by looking at the questionWords (for example you want to make sure that combination of some words are present in the user question) and use AIContext that provides additional info. As part of handling of user question, we determine what kind of command user wants to execute: get data from Costpoint or to execute some actions (other when to get data). You can call AIContext methods below to see what kind of command user is asking

  - boolean dataCommand();
  - boolean actionCommand();

  The return value from the **triggerFunction()** method tells framework what to do with this function:

  - 0 -- do not show function to AI
  - 1 -- show function to AI
  - -1 -- use triggerWords to see whether function is needed (default behavior)

  Example

```
@Override
public int triggerFunction(String[] questionWords,AIContext mgr){
    if(!mgr.actionCommand()||!mgr.haveRightsOnApp( "TMMTIMESHEET_APPROVE")){//mgr.isTeambotMode()||
        return 0;
    }
    //have to have approve and timesheet or something
    boolean approve=false;
    boolean time=false;
    for(String w:questionWords){
        switch(w){
        case "time":case "timesheet":case "timesheets":case "timecard":time=true;break;
        }
        if(action.equals(w)){
            approve=true;
        }
    }
    return (approve&&time)?1:0;
}
```

Abstract Methods

Method	Purpose	Required
checkParams()	Validate parameters before execution	Yes
invokeFunction()	Execute the main function logic	Yes
getFunctionEntity()	Return entity name for link generation	Yes

After the framework determines which functions to make available for AI to answer the user question, it will submit the user command to AI along with active functions. AI will then decide whether to return a response right away without calling any functions, it can also call one or several functions at the same time, or it can call several functions, analyze their return, and then call some extra functions. We do not control this behavior; we only need to execute functions and return their responses back to AI, so it can answer user questions based on that info. Potentially, developers can stop execution of the user command and return an error back to the user. In some rare cases, developers' function can stop execution and return a response back to the user instead of returning it to AI, more on that later. If, after submitting the user command to AI, AI decides to invoke one or several functions, the process goes as follows:

- The framework will loop over these functions and will invoke the **checkParams()** method first.
- If none of the functions returned an error back, the framework will proceed to the next step. It will invoke the method **invokeFunction(AIContext mgr, AIResponse res)** of each function. Again, the invoke function can return errors, in which case we will stop execution and will return an error directly back to the user.

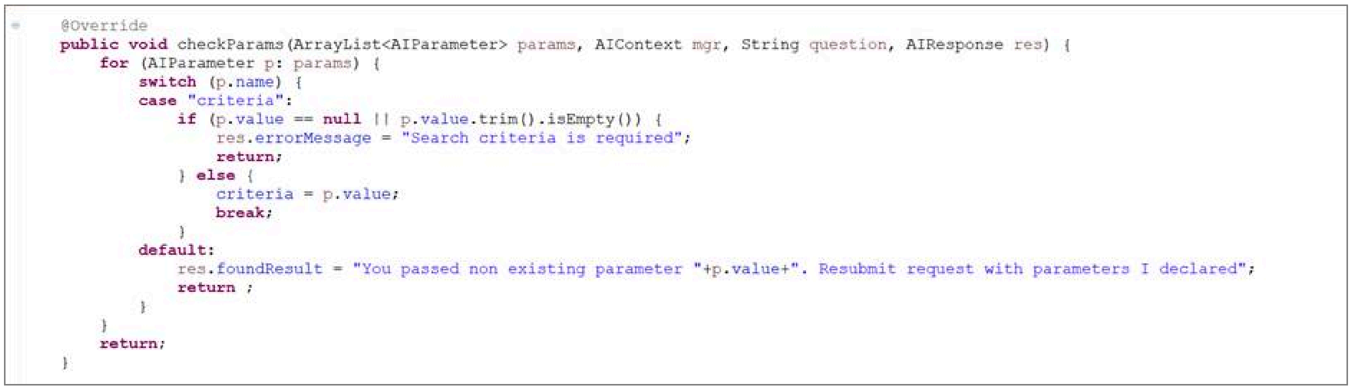
checkParams(ArrayList<AIParameter> params, AIContext mgr, String question, AIResponse res)

- Parameters
    - <**params**>: Parameters provided by AI model
    - <**mgr**>: AIContext for accessing framework services

   - <question>: Original user question for context
   - <res>: Response object to set results or errors
- Validates parameter values and types before function execution.
- Checks business logic constraints
- You can set **res.errorMessage** if validation fails, or set **res.foundResult** to skip the **invokeFunction()** call if desired.

You can use this opportunity to analyze function params, preserve them in instance variables, stop functions execution, if necessary, and return errors either to AI or to the user directly.

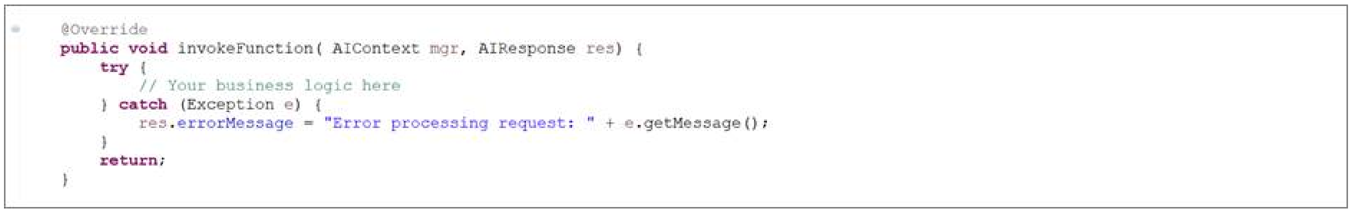
Example:



```
    @Override
    public void checkParams(ArrayList<AIParameter> params, AIContext mgr, String question, AIResponse res) {
        for (AIParameter p: params) {
            switch (p.name) {
            case "criteria":
                if (p.value == null || p.value.trim().isEmpty()) {
                    res.errorMessage = "Search criteria is required";
                    return;
                } else {
                    criteria = p.value;
                    break;
                }
            default:
                res.foundResult = "You passed non existing parameter "+p.value+". Resubmit request with parameters I declared";
                return ;
            }
        }
        return;
    }
```

invokeFunction(AIContext mgr, AIResponse res)

- Parameters
   - <mgr>: AIContext for accessing framework services and data
   - <res>: Response object to set execution results
- Executes the main function logic. In this method, you will perform the actual function work.
- You can set **res.foundResult** for successful completion, or set **res.errorMessage** for errors that should halt execution or set **res.askQuestion** to pause execution and ask user for input.



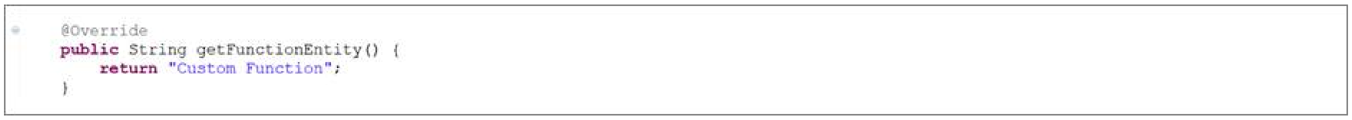
```
    @Override
    public void invokeFunction( AIContext mgr, AIResponse res) {
        try {
            // Your business logic here
        } catch (Exception e) {
            res.errorMessage = "Error processing request: " + e.getMessage();
        }
        return;
    }
```

At the end of this process, AI will generate final response that we will show to the user. In most cases, developers should also generate links to related applications as part of the **invokeFunction**() processing.

getFunctionEntity()

- As part of function definition you can implement **getFunctionEntity()** that will be used as a title in the application links we generate on client.
- It returns the entity name for link generation and categorizing function results. You can also use it for UI display purposes.
- It must return a non-null value.

Example:



```
@Override
public String getFunctionEntity() {
    return "Custom Function";
}
```

Listed below are additional common methods from AIFunction. The interface is constantly changing as we keep adding new functionalities. You must check the current interface definition and Java document from time to time.

- **adjustDefinition(AIContext mgr):** Dynamically modifies function definition at runtime. This allows ai functions to dynamically modify description based on things like user rights or any other info like client specific configuration, etc.
  Example:

  ```
  \@Override

  public void adjustDefinition(AIContext mgr){

         description = \"Invoke action to generate email.If specific \'on behal
  f\' or \'from\' instruction is provided, honor it at the end of the e-mail , ot
  herwise assume that email is from \"+mgr.getUserEmplName();               //AIF
  unctions will add <span class="underline">empl</span> name to the end of <span
  class="underline">funtion</span> description !

  }
  ```

- **isFileHandler():** Identifies functions that handle file/document processing. Automatically handled by the **AIFunctionFileHandler** subclass.

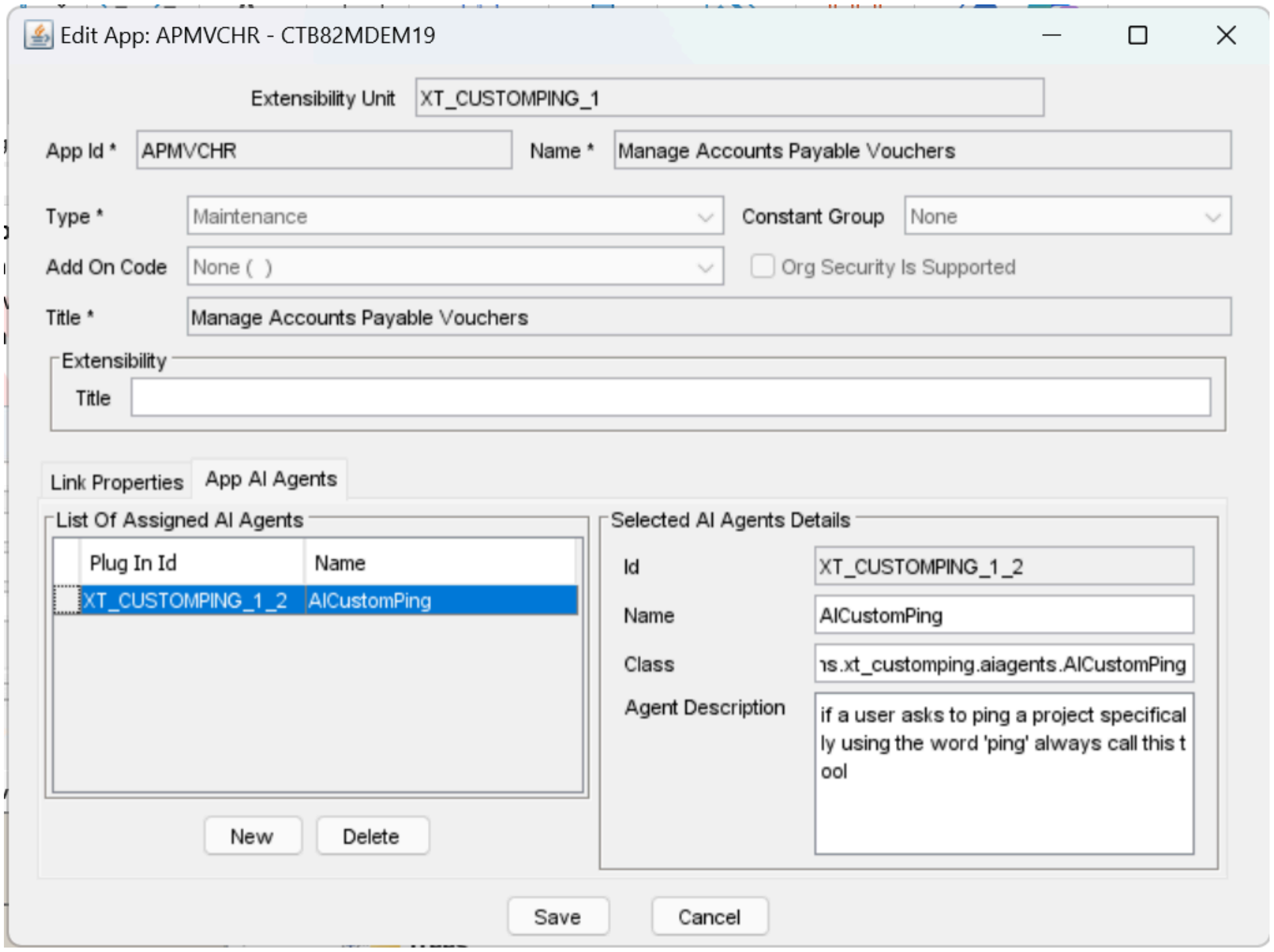
AI Agents Plug-in Registration

From the Framework perspective, it does not matter which app your AI plug-in will be registered in, so it is up to you to select the most logically related app to register your plug-in. There is a new tab in application definitions in the Designer that allows you to register your AI agents in the W_AI_AGENTS table, so the Framework knows that they exist.

1. In the Designer, select **Project » Unit » Apps**.

2. Select your application from the list. You can select either a new app you have created or an existing app that has been extended, then click **Edit**, this will open the Edit App screen.

3. Go to the App AI Agents tab, click **New** and enter required values in the fields of the **Selected AI Agent Details** group box:


Edit App: APMVCHR - CTB82MDEM19

Extensibility Unit	XT_CUSTOMPING_1
App Id *	APMVCHR
Name *	Manage Accounts Payable Vouchers
Type *	Maintenance
Constant Group	None
Add On Code	None ( )
Org Security Is Supported	☐
Title *	Manage Accounts Payable Vouchers
Extensibility Title	

Link Properties | App AI Agents

List Of Assigned AI Agents

Plug In Id	Name
XT_CUSTOMPING_1_2	AICustomPing

Selected AI Agents Details

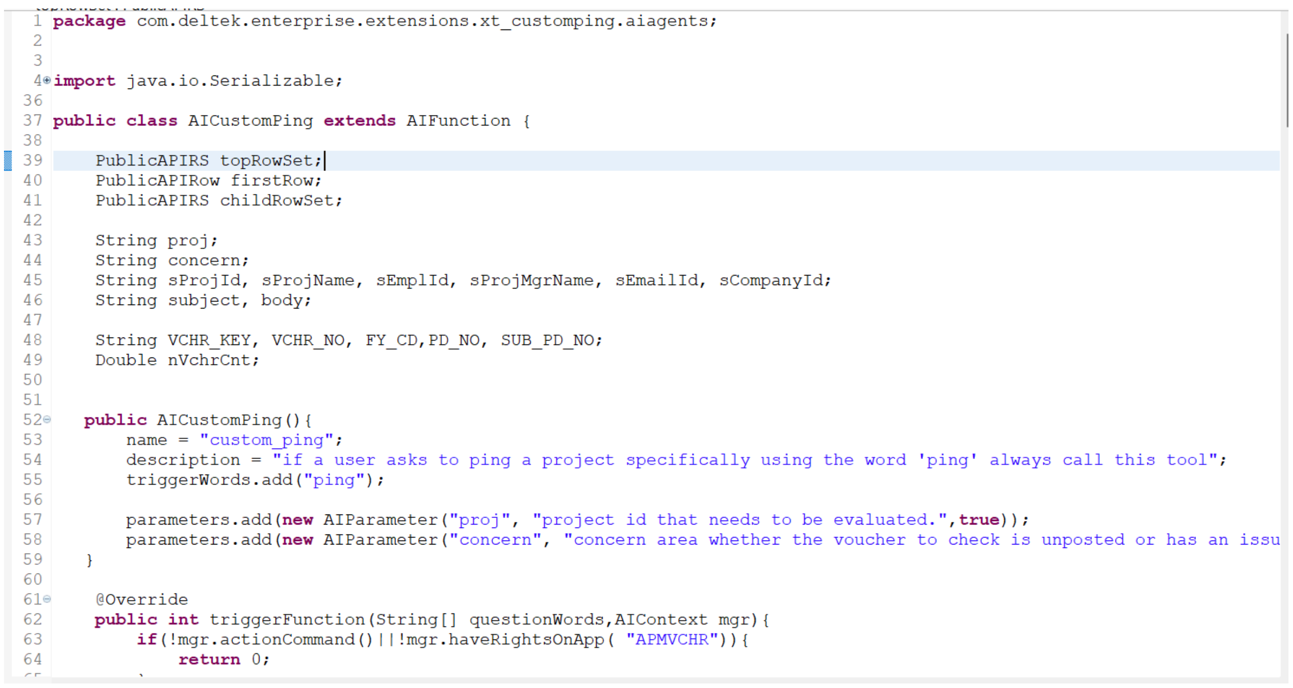
Id	XT_CUSTOMPING_1_2
Name	AICustomPing
Class	ıs.xt_customping.aiagents.AICustomPing
Agent Description	if a user asks to ping a project specifically using the word 'ping' always call this tool

The value you entered in the **Agent Description** field in the Designer will take precedence over the description value you entered in the AI function class. This will allow you to change the descriptions declaratively in the Designer and deploy the updates without the need to re-compile/deploy the Java class.

It is critical that descriptions entered in the Designer are always correct. We do, however, advise that as much as possible the description entered in both AI function Java classes and the Designer are in-sync to avoid potential problems. It is also critical for the Description (and trigger words) to be as specific as possible to your functionality, so it does not confuse AI. Description must not be too vague or generic. Otherwise, your agent will be called unnecessarily, will slow down answers to the easiest questions, and may provide incorrect answers to end users.

It is also important to review existing descriptions of AI plug-ins (both standard and Extensibility) to make sure your description does not logically interfere with other plug-ins. As of 2026.2, there is no such screen in the Ext Console to review all descriptions at once, but we will attempt to add one in the nearest feature.

In the sample below, the description value in the AI function in Java is also used in the **Agent Description** field in the Designer: "**if a user asks to ping a project specifically using the word 'ping' always call this tool.**"



```java
package com.deltek.enterprise.extensions.xt_customping.aiagents;


import java.io.Serializable;

public class AICustomPing extends AIFunction {

    PublicAPIRS topRowSet;
    PublicAPIRow firstRow;
    PublicAPIRS childRowSet;

    String proj;
    String concern;
    String sProjId, sProjName, sEmplId, sProjMgrName, sEmailId, sCompanyId;
    String subject, body;

    String VCHR_KEY, VCHR_NO, FY_CD,PD_NO, SUB_PD_NO;
    Double nVchrCnt;


   public AICustomPing(){
       name = "custom_ping";
       description = "if a user asks to ping a project specifically using the word 'ping' always call this tool";
       triggerWords.add("ping");

       parameters.add(new AIParameter("proj", "project id that needs to be evaluated.",true));
       parameters.add(new AIParameter("concern", "concern area whether the voucher to check is unposted or has an issu
   }

    @Override
    public int triggerFunction(String[] questionWords,AIContext mgr){
        if(!mgr.actionCommand()||!mgr.haveRightsOnApp( "APMVCHR")){
            return 0;
```

4. Click **Save**.

system.applicationinterface.ai.AIParameter

The AIParameter class is used to define parameters for AI functions and to receive parameter values when functions are invoked by the AI model. It extends **OpenAIParam** to provide the foundation for parameter handling in the Costpoint AI Framework. Each parameter defines its name, description, data type, and whether it's required, enabling the AI model to understand what inputs the function expects.

This class is used both to declare parameter(s) in the plugin AI function and also used by AI to pass a parameter value to **checkParams()** method. The parameter values will be stored either in **value** (string) or **objectValue** (JSON) variables.

Property	Type	Description
name	String	Parameter name

Property	Type	Description
description	String	Parameter description
required	boolean	Whether parameter is required
type	String	Parameter type (default: "string")
value	String	String parameter value
objectValue	JSONObject	Complex object parameter value

public AIParameter(String name, String description, boolean required)

- Parameters:
  - **name**: Parameter name (used by AI model to identify parameter)
  - **description**: Human-readable description explaining the parameter's purpose
  - **required**: Whether this parameter must be provided by AI model

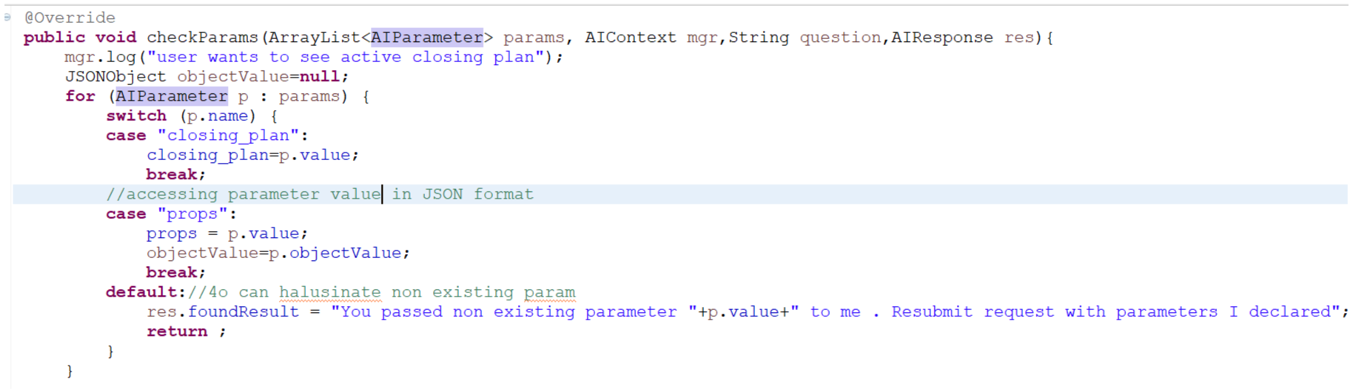
Example:

```
public AITaskGet() {
name = "...";
description =  "..."; //
triggerWords.add("...");
triggerWords.add("...");

// sample declaration, required closing plan parameter, to return in String forma
t.
parameters.add(new AIParameter("closing_plan", "Closing Plan to which user wants
to get tasks. Pass plan id or name if user mention specific closing plan, otherwi
se assume user want to know about current or open plan and pass 'current'.",tru
e));

//sample declaration, required multiple properties value, to return in JSON forma
t.
parameters.add(new AIParameter("props", "List of property/value in JSON format. B
y default assume user wants to get tasks currently assigned to the user but he ma
y also mention other properties. If user mentioned about status open pass status
as IP. Other properties are. Example: {'task':'1','job':'A','task_name':'B','prio
rity':'1','job':'E','status':'possible value NS,R,IP,W,C. Pass NS when status is
Not Started.Pass R when status is Ready.Pass IP when status is In-Process. Pass W
when status is Waiting, Pass C when status is Completed.','closing_day':'F','star
t_date':'MM/dd/yyyy','application':'G'...}",true));
}
```

When an AI function is invoked, the framework populates values in either 'value' (for simple types) or 'objectValue' (for complex types).

- Simple Types (string, number, boolean, integer)
  - Value is stored in 'value' variable as string
    `public String value;`
- Complex Types (arrays, objects)
  - Value is stored in 'objectValue' variable as 'JSONObject'
  - In some cases, we will pass it as parsed JSON object. This is for more advanced use cases where you tell AI specifically in parameter description to pass parameter value in JSON format.
    `public JSONObject objectValue;`

Example code in accessing the parameters values in checkParams() method:



```
@Override
public void checkParams(ArrayList<AIParameter> params, AIContext mgr,String question,AIResponse res){
    mgr.log("user wants to see active closing plan");
    JSONObject objectValue=null;
    for (AIParameter p : params) {
        switch (p.name) {
        case "closing_plan":
            closing_plan=p.value;
            break;
        //accessing parameter value in JSON format
        case "props":
            props = p.value;
            objectValue=p.objectValue;
            break;
        default://4o can halusinate non existing param
            res.foundResult = "You passed non existing parameter "+p.value+" to me . Resubmit request with parameters I declared";
            return ;
        }
    }
```

system.applicationinterface.ai.JSONObject

AIJSONObject is a utility class designed to represent and handle complex JSON structures within the Costpoint AI Framework. It provides a hierarchical data structure for parsing and accessing nested JSON objects, arrays, and simple properties. This class is primarily used by AIContext.parseComprexProps() and other framework methods to handle complex parameter structures passed from AI models.

Example:

```
AIJSONObject deptObj = mgr.parseComprexProps(jsonString);

// Used by AIContext.parseComprexProps()
public void invokeFunction(AIContext mgr, AIResponse res) {
    try {
        // Get complex JSON parameter
        String complexParam = getParameterValue("complex_data");

        // Parse into AIJSONObject structure
        AIJSONObject jsonObj = mgr.parseComprexProps(complexParam);

        // Access data using AIJSONObject properties
        processComplexData(jsonObj, res);

    } catch (DEException e) {
        res.errorMessage = "Failed to parse complex data: " + e.getMessage();
    }
}
```

system.applicationinterface.ai.AIResponse

AIResponse is the control object that AI functions use to communicate execution results, handle errors, response generation, manage user interactions, and control execution flow. It serves as the primary interface between AI functions and the framework, allowing functions to return data to the AI model, pause execution for user input, report errors, and manage complex dialog interactions.

It is available to use in AIFunction checkParams() and invokeFunction() methods. It has several properties that allow you to determine the result of function execution:

Property	Purpose	Usage
foundResult	Success result to return to AI, set when function executes successfully	If not null, framework assumes that function has been executed, and it will return this value back to AI as a return value of this function execution
errorMessage	Error message, set when fatal error occurs to halts execution	If not null, framework assumes that the function failed with error and will return the result back to user
askQuestion	Question to pause and ask user, set when user input if needed	If not null, framework assumes that it needs to pause execution of all functions and ask user a question defined by that variable, then resume functions execution based on user response. See more details on Dialogs.
generateLinks	Set to true, when you want to generate links on error	Rare use case for error handling, see more details below.

generateLinks: If true and if errorMessage is not null, the framework assumes that we deliberately stopped command execution to return errorMessage to the user, but we did it not because of error, but because we decided that returning the response back to AI is not needed. E.g., in case you want to return a large response back to the user as is and do not want to rely on AI to do it (it takes AI time to process/generate the final response, and AI tends to modify whatever function response you give to it). Since we assume we did not have an error, we will try to generate links to the user (see more on that later). In most cases, you should return the response back to AI and allow it to generate the response, as you generally do not know whether the user command was complex, and your function call could be a small part of overall user command execution.

Example:

```
res.foundResult = "Operation completed successfully";

res.errorMessage = "Invalid parameters provided";

res.askQuestion = "Do you want to continue with this operation?"+ AIContext.CONFI
RM_TEXT;

res.generateLinks = true;
```

Methods to ask user questions with dialogs:

- **askToEnterData(String headerText, AIField... fields)**: Asks user to fill out form fields using a structured dialog interface.
  - Parameters:
    - **headerText**: Header or instruction text displayed to user
    - **fields**: Variable number of `AIField` objects defining form fields

  Example:

```
if (mgr.canUseDialogs()) {

    AIField empId = new AIField();

    empId.objectId = \"EMPLOYEE_ID\";

    empId.label = \"Employee ID\";

    empId.required = \'Y\';

  empId.dataType = \'S\';



    AIField hours = new AIField();
    hours.objectId = \"HOURS\";

    hours.label = \"Hours Worked\";

    hours.required = \'Y\';

    hours.dataType = \'N\';



    res.askToEnterData(\"Please enter timesheet information:\", empId, hours);

} else {

// Fallback for older versions

   res.askQuestion = \"Please provide employee ID and hours worked (format: I
D,hours):\";

}
```

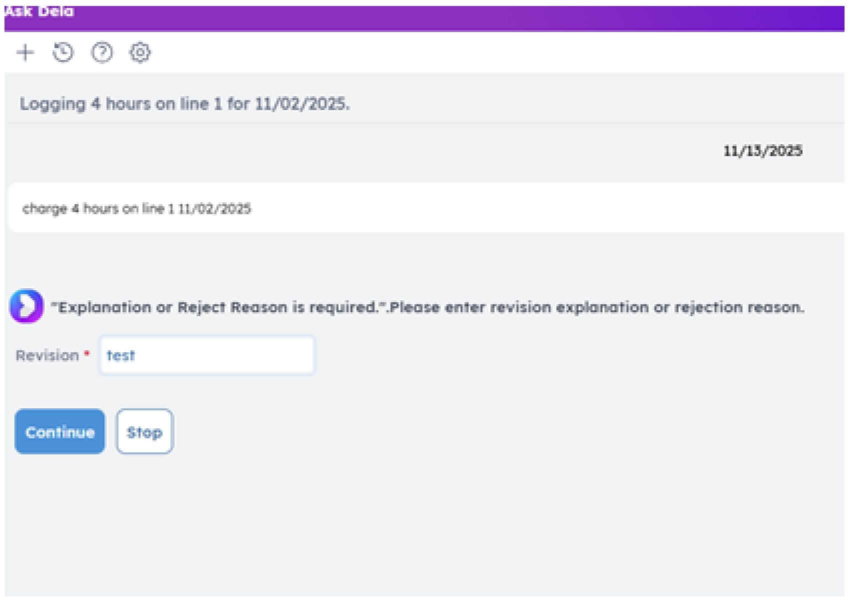
- **askToEnterData(String headerText, String[] buttonLabels, AIField... fields)**: Asks user to fill out form with custom button labels.
  - Parameters:
    - **headerText**: Header/instruction text
    - **buttonLabels:** Custom button labels (typically `["Submit", "Cancel"]`)
    - **fields**: Form field definitions

  Example:

```
combo.displayValues=t.toString();

combo.values=v.toString();

res.askToEnterData(m.getMessageText()+\".Please provide revision explanatio
n.\",new String[]{\"Continue\",\"Stop\"},combo);
```

Below are the typical use cases where you can use these methods.

1. If we need to ask the user to enter some data. E.g. Revision explanation that can be a text or a value from combo box.


Ask Dela

Logging 4 hours on line 1 for 11/02/2025.

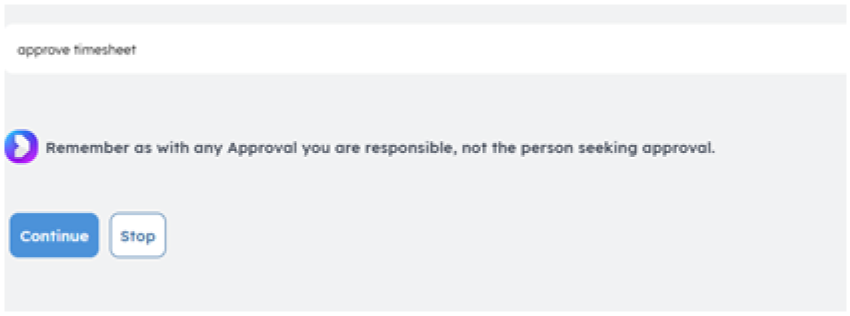
11/13/2025

charge 4 hours on line 1 11/02/2025

"Explanation or Reject Reason is required.".Please enter revision explanation or rejection reason.

Revision * test

2. If we need user to confirm something


approve timesheet

Remember as with any Approval you are responsible, not the person seeking approval.

system.applicationinterface.ai.AIFields

AIField is the class used to define form fields for user input collection in AI functions. It provides comprehensive support for various field types including text inputs, combo boxes, checkboxes, list boxes, and specialized formats like dates, emails, and URIs. Fields are used with AIResponse.askToEnterData() methods to create structured data collection dialogs in the Costpoint AI Framework.

Core Properties

- Use for field identity and content:

```
public String objectId = null; // Unique field identifier for value retrieval

public String defaultValue = null; // Default field value, usually string, exce
pt for multiselect comboboxes (list boxes) where it can be an array of strings,
eg ["Red","Green"]
```

```
public String label = null; // Field label/title displayed to user

public String description = null; // long text we associate with field, eg some
thing you show in tooltips/help text for the field or some instructions to user
on how to populate the field.
```

- Use to define field behavior:

```
public Character required = \'Y\'; // Y=Required, N=Optional

public Character dataType = \'S\'; // S=String, N=Number, I=Integer, D=Date,
B=Boolean
```

- MCP Validation Properties (Advanced):

```
public Integer maxLength = null;  // Maximum string length

public Integer minLength = null;  // Minimum string length

public String pattern = null;     // Regular expression pattern for string fiel
ds, you can define regex format eg \"\^[0-9]{10}\$\", or \"\^\\\\d{3}-\\\\d{2}-
\\\\d{4}\$\" for SSN

public String format = null;      // MCP formats: email, uri, date, date-time

public String maximum = null;     // Maximum numeric value

public String minimum = null;     // Minimum numeric value

public String multipleOf = null;  // Numeric multiple constraint

public String exclusiveMinimum;   // Exclusive minimum bound

public String exclusiveMaximum;   // Exclusive maximum bound
```

- Combo Box and List Box Properties:

```
public boolean multiSelect = false; // Allow multiple selections

public Integer maxItems = null;     // Maximum items for multi-select

public Integer minItems = null;     // Minimum items for multi-select

public String values = null;        // list of possible values for combo box. S
et them by calling setValues() method
```

```
public String displayValues = null; // list of possible display values for comb
o box. Set them by calling setDisplayValues() method
```

Methods

- **setValues(String... comboboxValues)**: Sets the internal values for combo box or list box options. Variable number of string values (no spaces, quotes, or pipe characters allowed)

Example:

```
AIField statusField = new AIField();

statusField.setValues(\"ACTIVE\", \"INACTIVE\", \"COMPLETED\");
```

- **setDisplayValues(String... comboboxDisplayValues)**: Sets the display labels for combo box or list box options. Variable number of display strings

Example:

```
AIField statusField = new AIField();

statusField.setValues(\"ACTIVE\", \"INACTIVE\", \"COMPLETED\");

statusField.setDisplayValues(\"Active Project\", \"Inactive Project\", \"Complete
d Project\");
```

Sample Implementation:

1. Text Fields

   - Basic Text Input

     ```
     AIField nameField = new AIField();
     nameField.objectId = "EMPLOYEE_NAME";
     nameField.label = "Employee Name";
     nameField.description = "Enter the full name of the employee";
     nameField.dataType = 'S';
     nameField.required = 'Y';
     ```

   - Text with Pattern Validation

     ```
     AIField phoneField = new AIField();
     ```

```
phoneField.objectId = "PHONE_NUMBER";
phoneField.label = "Phone Number";
phoneField.description = "Enter phone number (format: XXX-XXX-XXXX)";
phoneField.dataType = 'S';
phoneField.pattern = "^\\d{3}-\\d{3}-\\d{4}$";
phoneField.required = 'N';
```

2. Numeric Fields

   - Decimal Number Input

```
AIField salaryField = new AIField();
salaryField.objectId = "SALARY";
salaryField.label = "Annual Salary";
salaryField.description = "Enter annual salary amount";
salaryField.dataType = 'N';
salaryField.required = 'Y';
salaryField.minimum = "0";
salaryField.maximum = "999999.99";
```

   - Integer Input

```
AIField ageField = new AIField();
ageField.objectId = "AGE";
ageField.label = "Age";
ageField.description = "Enter age in years";
ageField.dataType = 'I';
ageField.required = 'Y';
ageField.minimum = "18";
ageField.maximum = "100";
```

3. Date Fields

   - Date Input (Special Format)

```
AIField startDateField = new AIField();
startDateField.objectId = "START_DATE";
startDateField.label = "Start Date";
startDateField.description = "Enter the project start date";
startDateField.dataType = 'S';     // String type
startDateField.format = "date";    // Special date format
startDateField.required = 'Y';
// Date format will be YYYY-MM-DD in client
```

- Date-Time Input

  ```
  AIField meetingTimeField = new AIField();
  meetingTimeField.objectId = "MEETING_TIME";
  meetingTimeField.label = "Meeting Time";
  meetingTimeField.description = "Enter meeting date and time";
  meetingTimeField.dataType = 'S';
  meetingTimeField.format = "date-time";
  meetingTimeField.required = 'Y';
  // Format: 2025-11-19T14:20:23Z
  ```

4. Boolean Fields (Checkboxes)

```
AIField activeField = new AIField();
activeField.objectId = "IS_ACTIVE";
activeField.label = "Active Employee";
activeField.description = "Check if employee is currently active";
activeField.dataType = 'B';
activeField.required = 'N';
activeField.defaultValue = "true";
```

5. Combo Boxes (Dropdown)

```
AIField departmentField = new AIField();
departmentField.objectId = "DEPARTMENT";
departmentField.label = "Department";
departmentField.dataType = 'S';
departmentField.required = 'Y';
// Internal values vs display names
departmentField.setValues("ENG", "MKT", "FIN", "HR");
departmentField.setDisplayValues("Engineering", "Marketing", "Finance", "Huma
n Resources");
```

6. List Boxes (Multi-Select)

```
AIField skillsField = new AIField();
skillsField.objectId = "SKILLS";
skillsField.label = "Skills";
skillsField.description = "Select all applicable skills";
skillsField.dataType = 'S';
skillsField.multiSelect = true;
skillsField.required = 'N';
```

```
skillsField.minItems = 1;
skillsField.maxItems = 5;
skillsField.setValues("JAVA", "PYTHON", "JAVASCRIPT", "SQL", "REACT");
skillsField.setDisplayValues("Java Programming", "Python", "JavaScript", "SQL
Database", "React Framework");
```

7. Specialized Format Fields

- Email Field

```
AIField emailField = new AIField();
emailField.objectId = "EMAIL";
emailField.label = "Email Address";
emailField.description = "Enter valid email address";
emailField.dataType = 'S';
emailField.format = "email";
emailField.required = 'Y';
emailField.maxLength = 100;
```

- URI Field

```
AIField websiteField = new AIField();
websiteField.objectId = "WEBSITE";
websiteField.label = "Website URL";
websiteField.description = "Enter company website URL";
websiteField.dataType = 'S';
websiteField.format = "uri";
websiteField.required = 'N';
```

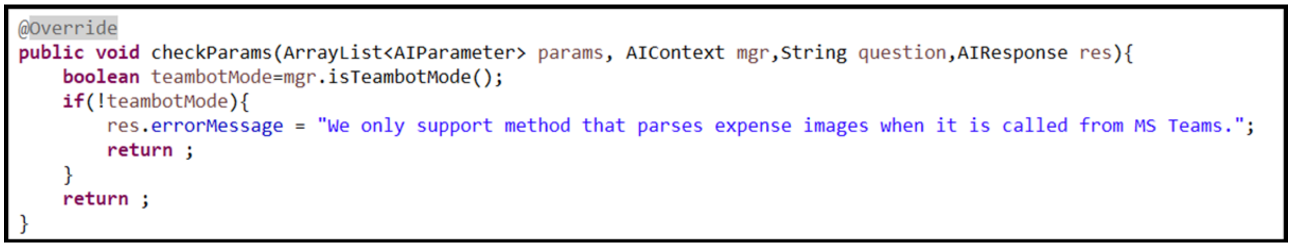
system.applicationinterface.ai.AIContext

AIContext interface is the core service provider for AI plugin development in the Costpoint AI Framework. It serves as the primary gateway for AI plugins to access system functionality, user information, data operations, and framework utilities. Every AI function receives an AIContext instance during execution, providing comprehensive access to the Costpoint ecosystem.

This object is available in AIFunction checkParams() and invokeFunction() methods. It provides a way for developers to get access to extra info about the command and various services the framework provides. Think of it as an object similar to App, Action, or ResultSet interface objects that the framework passes to regular developer plugins.

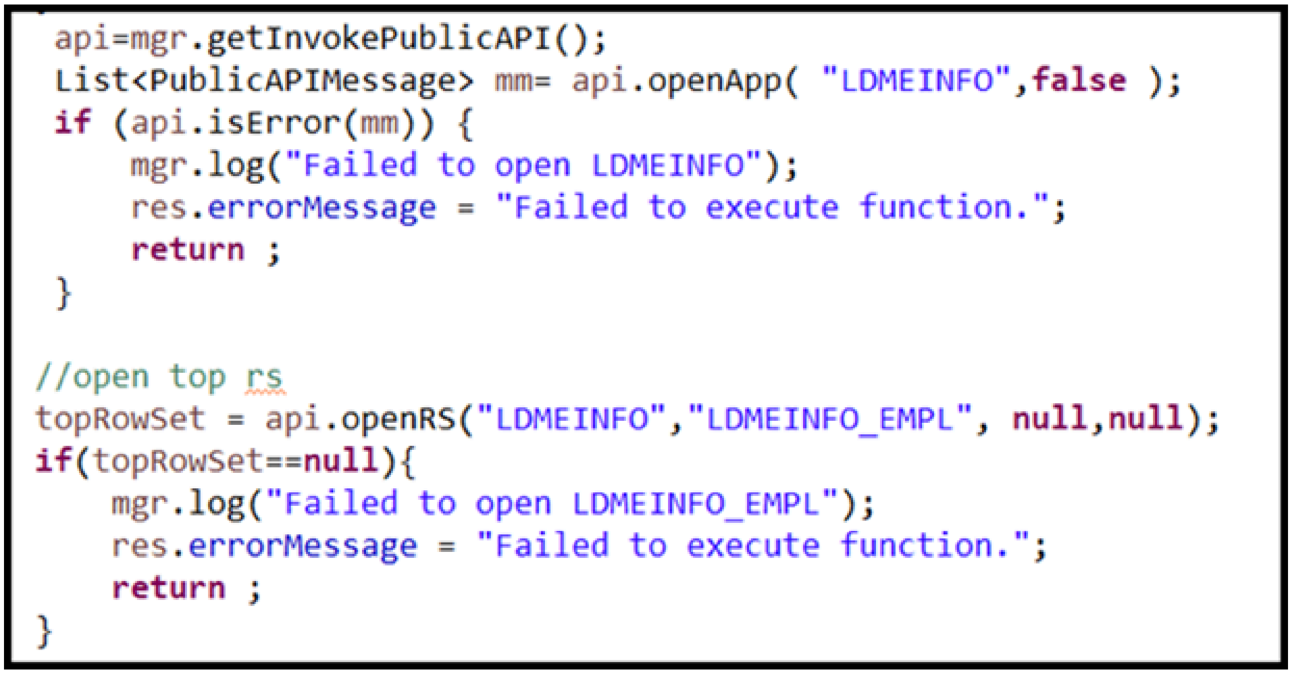
Listed below are some of the common methods from AIContext, but keep in mind the interface is constantly changing as we keep adding new functionality, so you should check the current interface definition and JavaDoc from time to time.

1. **isTeambotMode():** Returns true if the Ask Dela user question was invoked from Microsoft Teams, not from the browser client. This is important for determining the interaction context and adjusting responses accordingly.



```
@Override
public void checkParams(ArrayList<AIParameter> params, AIContext mgr,String question,AIResponse res){
    boolean teambotMode=mgr.isTeambotMode();
    if(!teambotMode){
        res.errorMessage = "We only support method that parses expense images when it is called from MS Teams.";
        return ;
    }
    return ;
}
```

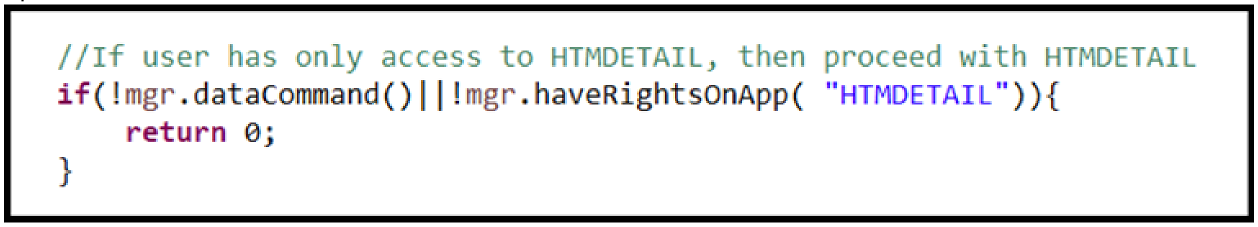
2. **log(String text):** allows you to log some useful info about function execution that we will log into AI logs. Note that it should be running in DEBUG mode to log.



```
 api=mgr.getInvokePublicAPI();
 List<PublicAPIMessage> mm= api.openApp( "LDMEINFO",false );
 if (api.isError(mm)) {
     mgr.log("Failed to open LDMEINFO");
     res.errorMessage = "Failed to execute function.";
     return ;
 }

//open top rs
topRowSet = api.openRS("LDMEINFO","LDMEINFO_EMPL", null,null);
if(topRowSet==null){
    mgr.log("Failed to open LDMEINFO_EMPL");
    res.errorMessage = "Failed to execute function.";
    return ;
}
```

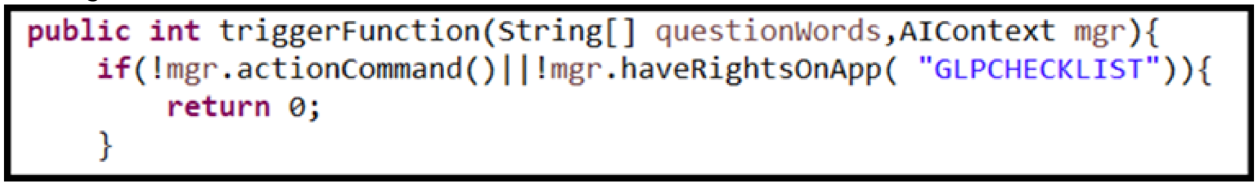
3. **dataCommand()** and **actionCommand():**

   - **dataCommand()** - Returns true when the system determines that the user wants to retrieve data. This is commonly used in triggerFunction() to determine if the AI function should activate for data retrieval operations.



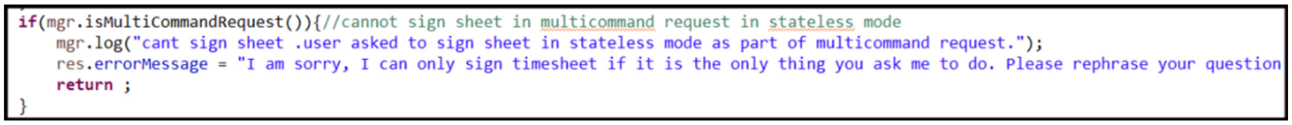
```
//If user has only access to HTMDETAIL, then proceed with HTMDETAIL
if(!mgr.dataCommand()||!mgr.haveRightsOnApp( "HTMDETAIL")){
    return 0;
}
```

- actionCommand() - Returns true when the system determines that the user wants to execute some action. This complements dataCommand() for action-oriented AI functions like creating, updating, or deleting records.



```
public int triggerFunction(String[] questionWords,AIContext mgr){
    if(!mgr.actionCommand()||!mgr.haveRightsOnApp( "GLPCHECKLIST")){
        return 0;
    }
```

4. isMultiCommandRequest(): Returns true if AI invoked several functions at the same time. Use this to return errors for functions that can only be invoked separately, or to change function execution behavior based on whether it was invoked alone or with other functions.



```
if(mgr.isMultiCommandRequest()){//cannot sign sheet in multicommand request in stateless mode
    mgr.log("cant sign sheet .user asked to sign sheet in stateless mode as part of multicommand request.");
    res.errorMessage = "I am sorry, I can only sign timesheet if it is the only thing you ask me to do. Please rephrase your question
    return ;
}
```

5. isTESSAdaptiveCardsEnabled(): Returns true when adaptive cards are enabled in the system.

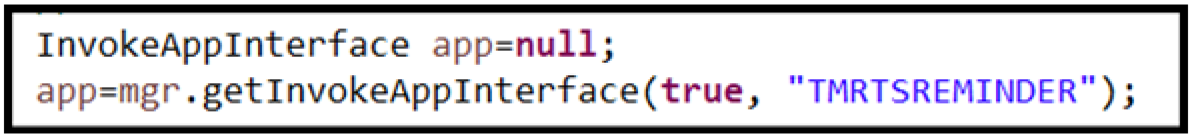
6. isUserAdaptiveCardsTeamsEnabled(): Returns true when the current user can receive adaptive cards in Teams.

7. isUserAdaptiveCardsEmailEnabled(): Returns true when the current user can receive email notifications.

8. getIICRInterface(): Gets the ICRInterface that uses AI to scan documents.

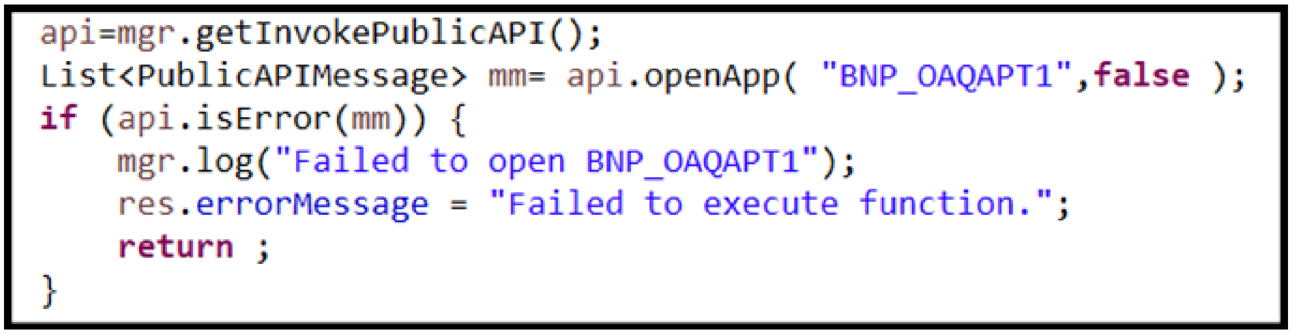
9. getContext(): Returns the application context object that some methods may require as a parameter.

10. getInvokeAppInterface(boolean allRights,String appId): Returns an InvokeAppInterface object for controlling Costpoint apps, such as opening apps and importing data. The allRights parameter should be passed as true when you need elevated permissions, although this needs to be discussed with FW if you believe you need to pass true.



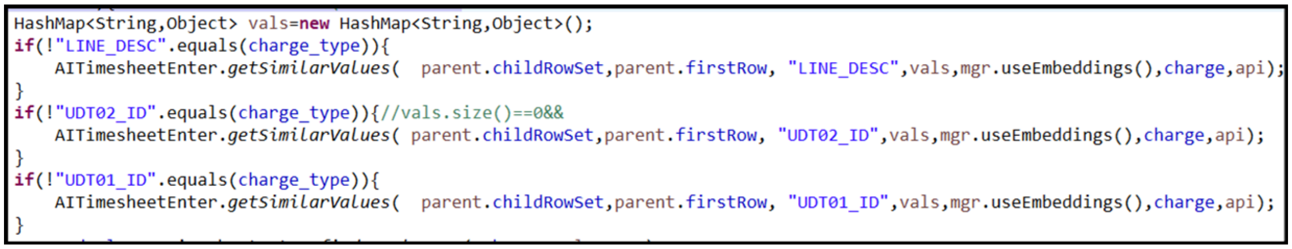
```
InvokeAppInterface app=null;
app=mgr.getInvokeAppInterface(true, "TMRTSREMINDER");
```

11. getInvokePublicAPI(): Returns a PublicAPIInterface object that provides low-level control over Costpoint apps. This is the most commonly used method for accessing application data and functionality.



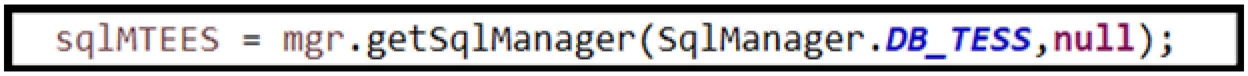
```
api=mgr.getInvokePublicAPI();
List<PublicAPIMessage> mm= api.openApp( "BNP_OAQAPT1",false );
if (api.isError(mm)) {
    mgr.log("Failed to open BNP_OAQAPT1");
    res.errorMessage = "Failed to execute function.";
    return ;
}
```

12. **useEmbeddings():** Returns true if embeddings service (semantic matching and search capabilities) is available.



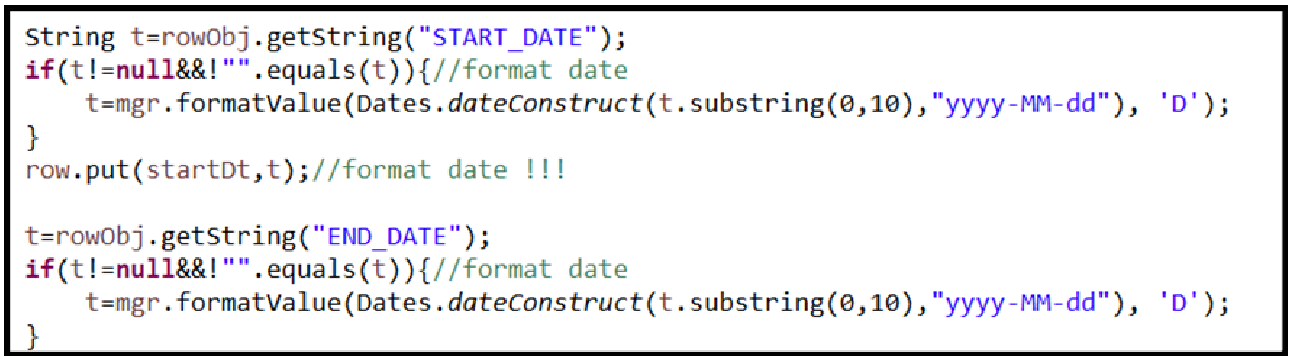
```
HashMap<String,Object> vals=new HashMap<String,Object>();
if(!"LINE_DESC".equals(charge_type)){
    AITimesheetEnter.getSimilarValues(  parent.childRowSet,parent.firstRow, "LINE_DESC",vals,mgr.useEmbeddings(),charge,api);
}
if(!"UDT02_ID".equals(charge_type)){//vals.size()==0&&
    AITimesheetEnter.getSimilarValues( parent.childRowSet,parent.firstRow, "UDT02_ID",vals,mgr.useEmbeddings(),charge,api);
}
if(!"UDT01_ID".equals(charge_type)){
    AITimesheetEnter.getSimilarValues(  parent.childRowSet,parent.firstRow, "UDT01_ID",vals,mgr.useEmbeddings(),charge,api);
}
```

13. **getSqlManager(String alias, Object objClass):** Gets a SQL manager for database operations. The alias parameter specifies the database connection (e.g., "DB_ADMIN", "DB_DATA").



```
sqlMTEES = mgr.getSqlManager(SqlManager.DB_TESS,null);
```

14. **formatValue(Object o,Character dataType):** Formats values based on their type using the default formatting algorithm. Data types include 'D' for dates, 'N' for numbers, and 'S' for strings.



```
String t=rowObj.getString("START_DATE");
if(t!=null&&!"".equals(t)){//format date
    t=mgr.formatValue(Dates.dateConstruct(t.substring(0,10),"yyyy-MM-dd"), 'D');
}
row.put(startDt,t);//format date !!!

t=rowObj.getString("END_DATE");
if(t!=null&&!"".equals(t)){//format date
    t=mgr.formatValue(Dates.dateConstruct(t.substring(0,10),"yyyy-MM-dd"), 'D');
}
```

15. **getUserEmplName():** Returns the current user's employee name.

16. **getUserEmplId()**: Returns the current user's employee ID.

17. **getUserId()**: Returns the current user's system user ID.

18. **getMyId()**: Returns a hash map of basic user properties including:

    - **user_id**: System user ID
    - **user_name**: User name
    - **employee_id**: Employee ID
    - **employee_name**: Employee name
    - **user_email**: User email address

    Example:

    ```
    HashMap<String,String> userInfo = mgr.getMyId();
    String email = userInfo.get("user_email");
    String empName = userInfo.get("employee_name");
    ```

    Note: Available in Costpoint 2025.4 and higher versions.

19. **getConfig()**: Gets reference to OpenAIConfig needed for creating InvokeAI instances.

20. **getGlobalConstant(String constId)**: Gets access to global constants that don't belong to specific applications.

    Example:

    ```
    Serializable version = mgr.getGlobalConstant("CP_VERSION");
    ```

    Note: Available in Costpoint 2025.4 and higher versions.

21. **parseProps(String value, Object objectValue)**: Parses function arguments defined as JSON objects into column/value maps.

22. **parseComprexProps(String parsedArgs)**: Parses more complex JSON objects into structured format.

23. **parseJSONMapProps(String props, Object objectValue)**: Parses JSON properties representing simple key:value maps.

    > **Note:** Available in Costpoint 2025.4 and higher versions.

24. **parseJSONArray(String props, Object objectValue)**: Parses JSON array arguments into ArrayList of strings.

    Example:

    `ArrayList<String> values = mgr.parseJSONArray(params.get(0).value, params.get(0).objectValue);`

    > **Note:** Available in Costpoint 2025.4 and higher versions.

25. **isValueID(String val)**: Simple method to determine if string looks like an ID rather than a name. It checks if value is uppercase and doesn't contain spaces.

    Example:

    ```
    if (mgr.isValueID(inputValue)) {
        // Treat as ID
    } else {
        // Treat as name, may need lookup
    }
    ```

    > **Note:** Available in Costpoint 2025.4 and higher versions.

26. **invokeActionByClient(String name, HashMap<String,String> parms)**: Passes information to browser for client-side action execution (currently used by call and email functions).

27. **getDateFormat()**:Returns current date format (usually MM/DD/YYYY).

28. **genReturnMessage(AIResponse resC)**: Generates proper return object for AI from manager responses.

    > **Note:** For function managers only.

29. **haveRightsOnApp(String appId)**: Checks if current user has any rights on the specified application.

    Example:

```
if (!mgr.haveRightsOnApp("GLQF")) {
    res.errorMessage = "Access denied to General Ledger application";
    return;
}
```

30. **domainRights()**: Returns list of domain IDs user has access to.

    Example:

```
HashSet<String> domains = mgr.domainRights();
if (!domains.contains("FINANCE")) {
    res.errorMessage = "Access denied to Finance domain";
    return;
}
```

> **Note:** Available in Costpoint 2025.4 and higher versions.

31. **findPersonIDByName(String nameOrID, String personType)**: Finds person ID by name, automatically detecting if input is already an ID. Returns a map of person IDs/names. If size > 1, user should choose from multiple matches.

    Example:

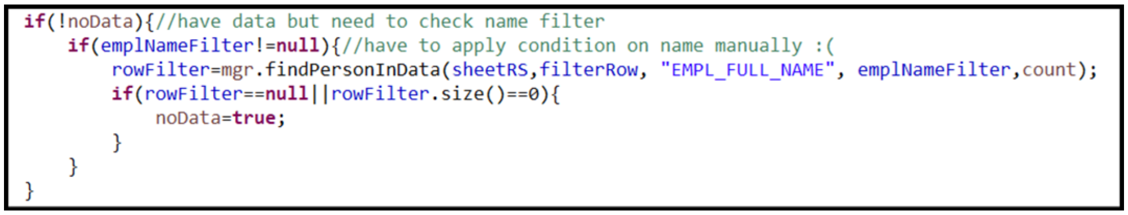
```
private String findEmployeeId(AIContext mgr, String employeeName) {
HashMap<String,String> persons = mgr.findPersonIDByName(employeeName, AIConte
xt.PERSON_EMPLOYEE);

    if (persons.size() == 0) {
        return null; // Not found
    } else if (persons.size() == 1) {
        return persons.keySet().iterator().next();
    } else {
        // Multiple matches - would need user selection logic
        return null;
    }
}
```

> Note: Available in Costpoint 2025.4 and higher versions.

32. **findSavedParam(String paramIdName, String appId, String rsId, String actionReportName)**: Finds saved parameters similar to the given parameter name in action/report result sets. Returns a map where keys are "paramId + paramName" and values are saved parameter IDs.

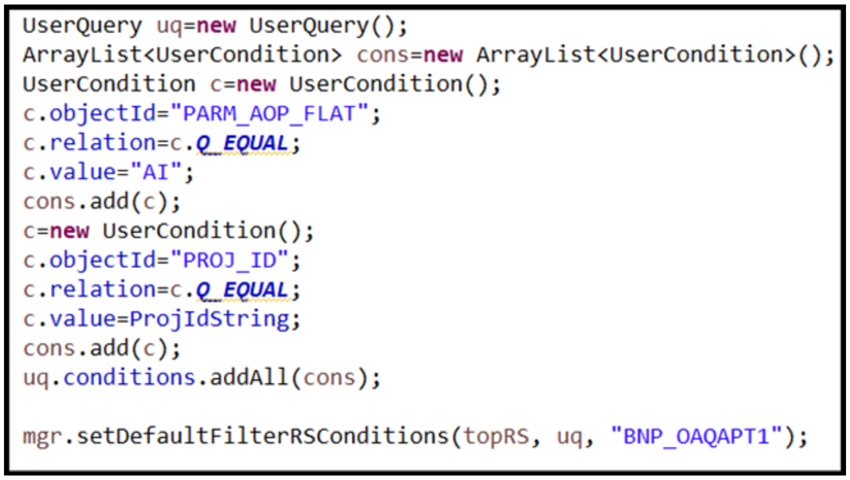
> Note:
>
> - Available in Costpoint 2025.4 and higher versions.
> - The Functions below assume you are using PublicAPIInterface . Since they are rather specific to the process of handling AI function execution, we put them here instead of putting them directly into PublicAPIInterface

33. **findPersonInData(PublicAPIRS rs,PublicAPIRow parentRow,String columnName,String value,int count)**: A convenience method to search for person in particular column based on given value. Given this is a common task, framework provides a default implementation of such algorithm.



```
if(!noData){//have data but need to check name filter
    if(emplNameFilter!=null){//have to apply condition on name manually :(
        rowFilter=mgr.findPersonInData(sheetRS,filterRow, "EMPL_FULL_NAME", emplNameFilter,count);
        if(rowFilter==null||rowFilter.size()==0){
            noData=true;
        }
    }
}
```

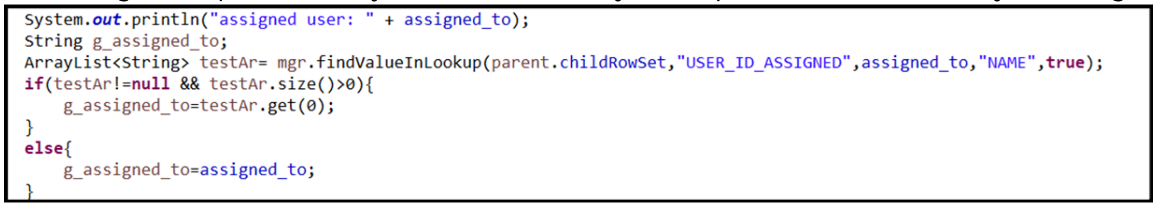
34. **setDefaultFilterRSConditions(PublicAPIRS rs ,UserQuery q,String fakeAppId)**: In some cases, you want to clear filter columns after populating them by applying filter object default values. This method helps manage default filter conditions on result sets.



```
UserQuery uq=new UserQuery();
ArrayList<UserCondition> cons=new ArrayList<UserCondition>();
UserCondition c=new UserCondition();
c.objectId="PARM_AOP_FLAT";
c.relation=c.Q_EQUAL;
c.value="AI";
cons.add(c);
c=new UserCondition();
c.objectId="PROJ_ID";
c.relation=c.Q_EQUAL;
c.value=ProjIdString;
cons.add(c);
uq.conditions.addAll(cons);

mgr.setDefaultFilterRSConditions(topRS, uq, "BNP_OAQAPT1");
```

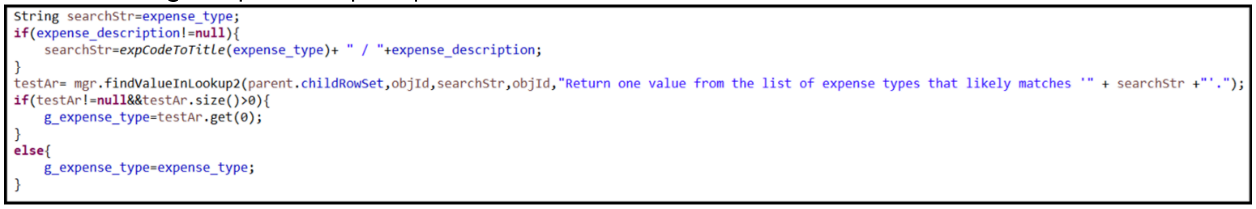
35. **findValueInLookup** and **findValueInLookup2**:

- **findValueInLookup(PublicAPIRS rs, String objectId, String value, String lkpNameCol,Boolean trySimilar)**: A convenience method to find values in lookup tables. This is frequently used for validating and converting user input to valid system values. The trySimilar parameter enables fuzzy matching.



```
System.out.println("assigned user: " + assigned_to);
String g_assigned_to;
ArrayList<String> testAr= mgr.findValueInLookup(parent.childRowSet,"USER_ID_ASSIGNED",assigned_to,"NAME",true);
if(testAr!=null && testAr.size()>0){
    g_assigned_to=testAr.get(0);
}
else{
    g_assigned_to=assigned_to;
}
```

- **findValueInLookup2(PublicAPIRS rs, String objectId, String value, String lkpNameCol, String prompt)**: An enhanced version of findValueInLookup that accepts a custom prompt for AI-assisted lookup. This method can use AI to help match similar values. If an exact match is not found, it uses AI to choose 'similar value' given provided prompt.



```
String searchStr=expense_type;
if(expense_description!=null){
    searchStr=expCodeToTitle(expense_type)+ " / "+expense_description;
}
testAr= mgr.findValueInLookup2(parent.childRowSet,objId,searchStr,objId,"Return one value from the list of expense types that likely matches '" + searchStr +"'.");
if(testAr!=null&&testAr.size()>0){
    g_expense_type=testAr.get(0);
}
else{
    g_expense_type=expense_type;
}
```

App Links

The framework provides methods in the AI context for generating links to related applications in various scenarios. When user clicks on these generated links, it will open application in context by executing query on top result set based on conditions passed UserQuery parameter.

- **genTopRSLink(PublicAPIRS topRS,UserQuery q)**: This is the simpler version that generates a basic AI link to an application. It will use default algorithm to come up with link title.
- **genTopRSLink(PublicAPIRS topRS, UserQuery q, String linkTitle)**: An improved version that generates AI link to an application with custom title.

Note: Available in Costpoint 2025.4 and higher versions.

- **genTopRSLink(PublicAPIRS topRS,UserQuery q,String linkTitle,ArrayList<String> linkIds,PublicAPIRow row)**: This method allows you to customize link title and print values of columns passed by linkIds parameter taken from passed row object.

Parameters	Purpose
PublicAPIRS topRS	The result set to generate a link for
UserQuery q	User query context for opening the app
String linkTitle	Custom title for the generated link
ArrayList<String> linkIds	List of column IDs to include in the link data

Parameters	Purpose
PublicAPIRow row	Specific row data to extract values from

Dialogs

This explanation describes a question-and-answer mechanism. In some cases, additional input from the user is needed to complete or execute the function. The framework provides a way to do it, but it requires some effort from developers because we cannot return a response from the user right away. You can pause function execution if you set the AIResponse object askQuestion property to a not-null value inside your invokeFunction() method. You cannot ask questions from the checkParams() method. After invoking your invokeFunction() method, the framework checks whether askQuestion is not null and, if so, it pauses execution of all other functions and returns the askQuestion string response back to the user. The user is expected to reply to the askQuestion. The framework will send that reply back to the server. The server will resume function execution by calling the paused function's invokeFunction() method again (and then invokeFunction() of other functions if we have many). You can pause execution several times. But you have to remember that, since we stopped function execution and then resumed it by calling the same invokeFunction(), it is the developer's responsibility to preserve all needed state info in the function instance variables (the framework will keep function objects alive, so your state data will be preserved) and be able to resume function execution from the last "paused place."

To get access to the user response to askQuestion, AIContext object provides several methods:

- isConfirmOK(), isConfirmCancel(): In case you expect user just to confirm something, you can use these convenience methods that will ok at user reply to determine whether it looks like Ok or Cancel (user can reply several ways so it makes sense to use these functions instead of analyzing response manually)
- getUserReply(): In case you expect user to provide you with some extra information, you can call this method to get user response as is, you will be responsible for making sense out of it.
- canUseDialogs(): Checks if the dialog functionality (`getUserReplyValues`, `askToEnterData`) is available.

  Note: Available in Costpoint 2026.1 and higher versions.

- getUserReplyValues(): Analyzes response from client when using `AIResponse.askToEnterData()`.
  Example:

```
if (mgr.canUseDialogs()) {
    HashMap<String,String> values = mgr.getUserReplyValues();
    if (values != null) {
        String fieldValue = values.get("fieldObjectId");
    } else {
        // User cancelled
    }
}
```

Note:

- Available in Costpoint 2026.1 and higher versions.
- Check first with canUseDialogs()

Manual Plugin Execution

You can have a use case when you already have an existing plugin that you would want to call from another plugin to reuse the code. This is a rather rare use case and requires careful handling.

To execute another plugin manually, you need to call the AIContext method:

- AIManualFunctionState invokeFunctionManually(String funcId,HashMap<String,String> args,AIResponse res): Manually invokes another AI plugin to reuse its functionality. Return state object to check if function completed or paused for user input.

To resume plugin execution in case the plugin asks user a question, you need to call the method:

- AIManualFunctionState resumeFunctionManually(AIManualFunctionState state)
  This is a special object that allows you to determine plugin execution status. It currently has a single method, isFunctionPaused(), that returns true if a manually executed plugin paused execution to ask the user a question.
  In case isFunctionPaused() returns true, you are supposed to preserve AIManualFunctionState in an instance variable and return control back to the framework. The framework will ask the user a question and then it will call your invokeFunction() method again. If your AIManualFunctionState is not null, you are supposed to invoke resumeFunctionManually(AIManualFunctionState state) method and pass your state to it. It will return a new state back. It could be in isFunctionPaused() state again, in which case you have to save the state and repeat dialog logic. Otherwise, you can assume that the plugin executed completely, and you can set the state to null and exit.
  Example:

```
// Initial invocation
HashMap<String,String> args = new HashMap<>();
args.put("param1", "value1");
AIManualFunctionState state = mgr.invokeFunctionManually("other_plugin", args,
res);
if (state.isFunctionPaused()) {
    // Function paused for user input
    // Later, after getting user response:
    AIManualFunctionState resumedState = mgr.resumeFunctionManually(state);
}
```

system.applicationinterface.ai.AIFunctionFileHandler

AIFunctionFileHandler is a specialized abstract base class that extends **AIFunction** to handle file and document processing in AI. It enables AI functions to process files (images, PDFs, and other documents) that users upload or send through Microsoft Teams or Ask Dela UI in Costpoint. The framework automatically detects when users send files and routes them to appropriate file handler functions based on file type and content analysis.

When Ask Dela is used, user can upload files into Costpoint and we provide a way to automatically handle these files if you extend AIFunctionFileHandler. AIFunctionFileHandler extends the AIFunction class and provides specific file handler methods. Technically, these handlers are not AI plugins because AI does not invoke them -- we invoke them manually, but they are exposed to developers as plugins and serve similar purposes (do AI things), so we cover them in the same document.

The way the process works: user uploads a file either using Ask Dela UI in CP or Teams, the framework sends this file to the server. The file gets analyzed to determine whether the file is an image (usually a PDF file with an image inside) or the file is a structured PDF file (usually an editable structured PDF format that represents some standard editable document or form).

In case we determine that the file is an image, we loop over all plugins that extend AIFunctionFileHandler and call two methods:

- **getImageFileDescr()**: This method should return some short constant in upper case, e.g. RECEIPT
- **getImageDescPrompt()**: This method should return a sentence that will be sent to AI, e.g. : Return RECEIPT if data represents a receipt, ticket or expense explanation that can be used in expense reports.

Based on the collected data, we will submit a special request to AI, along with the image, and ask it to classify the image into one of the categories defined by constants returned by **getImageFileDescr()**, and we will pass text from **getImageDescPrompt()** to help AI decide how to classify the image.

For example, we may have 2 plugins: an expense report plugin that takes a user receipt and automatically creates an expense out of it, and you may have a contract plugin that takes an image and automatically generates a contract from it. The expense plugin will return RECEIPT and the contract plugin will return CONTRACT constants, and they will return corresponding prompts to help AI classify the image.

If AI returns back one of the constants that correspond to AIFunctionFileHandler plugins, the framework will assume that that plugin will be able to handle the image.

Similarly, if the document is an editable PDF, the framework will loop over AIFunctionFileHandler plugins and will invoke method:

- **canHandlePDF(PDDocument doc)** - First plugin that returns true will be used to handle that document. After we determine which plugin can handle the document, we will invoke it, similar to the regular AIFunction plugin.

First, we call the method:

- storeDocument(String fileName, String mimeType, ArrayList<String> images, PDDocument pdfData, byte[] docBytes)

So that the plugin can save document data in the instance variables.

Then we call the **checkParams()** method and then we call the **invokeFunction()** method. You will be able to pause execution of plugin and ask user question by setting **askQuestion** property. As with regular plugins, framework will resume execution by calling **invokeFunction()** method again and you can get user response the same way as you do it from regular plugin. You can also stop execution by setting **errorMessage** or return normal response to user by setting **foundResult**.

Example:

```
// example AI function plug-ins that extend the AIFunctionFileHandler

public class AIContractOCR extends AIFunctionFileHandler {

private ArrayList\<String\> images;

private String fileName;

private byte[] fileContent;

public AIContractOCR() {

name = \"addContractOCR\";

description = \"special function to add contract based on image or pdf file from
teambot.\"; //

}

\@Override

public void storeDocument(String fileName, String mimeType, ArrayList\<String\> i
mages, PDDocument pdfData, byte[] docBytes) {

this.fileContent = docBytes;

this.images = images;

this.fileName = fileName;
this.mimeType = mimeType;

this.pdfData = pdfData;

}

\@Override

public String getImageFileDescr() {

return \"CONTRACT\";

}

\@Override

public String getImageDescPrompt() {

return \"Return CONTRACT if image appears to represent one the of the forms: STANDARD FORM 26, STANDARD FORM 1449, DD FORM 1155, STANDARD FORM 33.\";

}


\@Override

public boolean canHandlePDF(PDDocument doc) {

//analyze pdf to see whether it looks like contract - TBD

PDAcroForm acroForm = doc.getDocumentCatalog().getAcroForm();

List\<PDField\> fields = null;

if (acroForm==null\|\|(fields=acroForm.getFields())==null\|\|fields.size()==0 {

return false;

}

//ADD extra logic here you can parse doc and store its structure
return true;

}

}
```

Sample Implementation

Customized Ping

Requirements: Create a custom Dela agent that will allow a power user responsible for financial close to reach out (ping) product managers responsible for projects with "pending" unposted vouchers. This customization will allow for streamlined process for notifying project managers instead of manually figuring who leads each project and crafting/sending an email.

Implementation:

- Register new custom Dela agent through Ext Console -- e.g. "Custom Ping"
- The agent should do the following:
  If a user provides a command like: "Ping project 1003", "Ping for unposted vouchers in project 1003", "Ping project 1003 for voucher issues" the agent will:
  Receive two parameters:
  - Project ID (required).
  - Problem area (optional). Special area of concern (if provided in the user request) e.g. "voucher issues" or "unposted vouchers"
- Suggested AI function description: "if a user asks to ping a project specifically using the word "ping" always call this tool".

AI function logic:

- Agent should find project manager for project 1003.
- Find e-mail for the project manager
- send e-mail to the project manager with fixed text:
  ```
  "Hello, friendly reminder to post all pending vouchers for your project:
  ID, Name
  Our company's ability to close books depends on your attention and diligence in
  this matter.
  Thank you,
  Office of CFO"
  ```

Suggested Code

```
public class AICustomPing extends AIFunction {

    PublicAPIRS topRowSet;

    PublicAPIRow firstRow;

    PublicAPIRS childRowSet;

    String proj;

    String concern;

    String sProjId, sProjName, sEmplId, sProjMgrName, sEmailId, sCompanyId;

    String subject, body;

    String VCHR_KEY, VCHR_NO, FY_CD,PD_NO, SUB_PD_NO;

    Double nVchrCnt;

   public AICustomPing(){

       name = \"custom_ping\";

       description = \"if a user asks to ping a project specifically using the wo
rd \'ping\' always call this tool\";

       triggerWords.add(\"ping\");



//added a required 1st param for Proj and an optional 2nd param for concerned are
a.

       parameters.add(new AIParameter(\"proj\", \"project id that needs to be eva
luated.\",true));

       parameters.add(new AIParameter(\"concern\", \"concern area whether the vou
cher to check is unposted or has an issue, in case user didn\'t specify dafault w
ill be unposted}\",false));

   }
```

```
//implement a triggerFunction that will check if the user prompt/question is an a
ction command and if user has access rights to APMVCHR.

    \@Override

public int triggerFunction(String[] questionWords,AIContext mgr){

        if(!mgr.actionCommand()\|\|!mgr.haveRightsOnApp( \"APMVCHR\")){

            return 0;

        }

        return -1;

    }

    \@Override

    public void checkParams(ArrayList\<AIParameter\> params, AIContext mgr,String
question,AIResponse res)  {

        mgr.log(\"user wants to ping the project manager for unposted vouchers or
vouchers with issue\");

//prepare and validate param, then save to variables.

        for (AIParameter p : params) {

            switch (p.name) {

            case \"proj\":

                proj=p.value;

                break;

            case \"concern\":
                concern=p.value;

                break;

            default://4o can hallucinate non existing param

                res.foundResult = \"You passed non existing parameter \"+p.valu
e+\" to me. Resubmit request with parameters I declared\";

                return ;

            }

        }

        if (proj == null) {

            res.errorMessage = \"Please provide project\";

            return;

        }



        //retrieve email assigned to Project Manager.

        getProjMgrEmail (mgr,proj);

        if (sProjName == null \|\| \"\".equals(sProjName.trim())) {

            //display error that project doesn\'t exists

            res.errorMessage = \"Invalid Project.\";

            return ;

        }



        if (sEmailId == null \|\| \"\".equals(sEmailId.trim())) {
            //display error is email id is not setup for the project manager</spa
n>

            res.errorMessage = \"Error retrieving project manager email assigned
to the project. Please ensure they are properly setup in Projects and Employee ta
bles.\";

            return ;

        }

        return ;

    }



    \@Override

    public void invokeFunction( AIContext mgr,AIResponse res) {

        PublicAPIInterface api = null;

        try{

          //invoke Public APIs to open application

            api=mgr.getInvokePublicAPI();

            List\<PublicAPIMessage\> mm= api.openApp( \"APMVCHR\",false );

            if (api.isError(mm)) {

                mgr.log(\"Failed to open APMVCHR\");

                res.errorMessage = \"Failed to open Manage Voucher screen. Perhap
s user does not have rights.\";

                return ;

            }
```

```
//open top rs

            topRowSet = api.openRS(\"APMVCHR\",\"APMVCHR_VCHRHDR\", null,null);

            if(topRowSet==null){

                mgr.log(\"Failed to open APMVCHR_VCHRHDR\");

                res.errorMessage = \"Failed to open Manange Voucher screen. Perha
ps user does not have rights.\";

                return ;

            }

//check if there are unposted vouchers with transactions for the given project. Y
ou can further enhance this logic once you started using the "concern" parameter.
You can conditionally retrieve either unposted vouchers or voucher with issues de
pending on the user prompt.

            if(proj!=null){ //query all vouchers with transactions for the given
PROJ_ID

                // get all the list of vchr_key where vouchers contains transacti
on line for the given PROJ_ID

                getVchrList(mgr,proj);

                if (nVchrCnt==0) {

                    res.foundResult = \"No pending voucher for posting, no need t
o ping the project manager.\";

                    return;

                }

            }
            //send email to Proj Manager

            subject = \"Pending Vouchers for Posting\";

            body = \"Hello \" +sProjMgrName + \", friendly reminder to post all p
ending vouchers for your project: \"

                    + \"\\n \" +proj + \" \" + sProjName

                    + \"\\n Our company\'s ability to close books depends on your
attention and diligence in this matter.\"

                    + \"\\n\\n                Thank you,\"

                    + \"\\n                  Office of CFO\";



            String sEmail = \"An email has been sent to \"+sEmailId +\" with the
following message: \"

                    + \"\\n Subject: Pending Vouchers for Posting\"

                    + \"\\n\" + body;



            HashMap\<String,String\> parms=new HashMap\<String,String\>();

            parms.put(\"address\",sEmailId );

            parms.put(\"subject\",subject );

            parms.put(\"body\",body );

            mgr.invokeActionByClient(\"email\", parms);

            res.foundResult = \"Ok. Send the following message as is, but format
it to look like an email, \" + sEmail;

        }
        catch(Throwable ex){

            mgr.log(ex.getMessage()+\" \"+ IntegrationUtils.getStackTrace(ex));

            res.errorMessage = \"Failed to get voucher lists.\";

            return ;

        }

        finally{

            if(topRowSet!=null&&api!=null){

                try {

                    api.closeApp(\"APMVCHR\");

                } catch (DEException e) {

                    mgr.log(e.getMessage()+\" \"+ IntegrationUtils.getStackTrac
e(e));

                    res.errorMessage = \"Failed to get voucher lists.\";

                }

            }

        }

        return ;

    }



    private void getProjMgrEmail(AIContext mgr, String proj) {

        SqlManager sqlMgrData = null;

        try {
            sqlMgrData = mgr.getSqlManager(SqlManager.DB_DATA, null);

            String sSql = \"SELECT P.PROJ_NAME, P.EMPL_ID,P.PROJ_MGR_NAME,E.EMAI
L_ID \"

                    + \" FROM PROJ P, EMPL E \"

                    + \" WHERE P.EMPL_ID = E.EMPL_ID AND P.COMPANY_ID = E.COMPAN
Y_ID \"

                    + \" AND P.COMPANY_ID = :sCompanyId \"

                    + \" AND P.PROJ_ID = :sProjId \"

                    + \" INTO :sProjName, :sEmplId, :sProjMgrName, :sEmailId\";

            HashMap\<String, Serializable\> binds = new HashMap\<\>();

            binds.put(\"sProjId\", proj);

            binds.put(\"sEmplId\", null);

            binds.put(\"sProjMgrName\", null);

            binds.put(\"sEmailId\", null);

            binds.put(\"sProjName\", null);

            binds.put(\"sCompanyId\", mgr.getGlobalConstant(\"CP_COMPANY_ID\"));

            sqlMgrData.setBindsTable(binds);

            sqlMgrData.SqlExecuteQuery(sSql);



            sEmailId = (String) binds.get(\"sEmailId\");

            sProjMgrName = (String) binds.get(\"sProjMgrName\");

            sProjName = (String) binds.get(\"sProjName\");
```

```
        } catch (SQLException e) {

            mgr.log(ExceptionUtils.getStackTrace(e));

        }

        finally{

            if (sqlMgrData != null)

                sqlMgrData.close();

        }

    }

    private void getVchrList(AIContext mgr, String proj) {

        SqlManager sqlMgrData = null;

        try {

            sqlMgrData = mgr.getSqlManager(SqlManager.DB_DATA, null);

            String sSql = \"SELECT COUNT(H.VCHR_KEY) \"

                    + \" FROM VCHR_LN_ACCT L, VCHR_HDR H \"

                    + \" WHERE H.VCHR_KEY = L.VCHR_KEY AND H.S_VCHR_TYPE = \'AP\'
AND L.PROJ_ID = :sProjId \"

                    + \" INTO :nVchrCnt \";



            HashMap\<String, Serializable\> binds = new HashMap\<\>();

            binds.put(\"sProjId\", proj);
            binds.put(\"nVchrCnt\", null);

            sqlMgrData.setBindsTable(binds);

            sqlMgrData.SqlExecuteQuery(sSql);



            nVchrCnt = (Double)binds.get(\"nVchrCnt\");

        } catch (SQLException e) {

            mgr.log(ExceptionUtils.getStackTrace(e));

        }

        finally{

            if (sqlMgrData != null)

                sqlMgrData.close();

        }

    }




    \@Override

    public String getFunctionEntity() {

        return \"Unposted AP Vouchers\";

    }

}
```