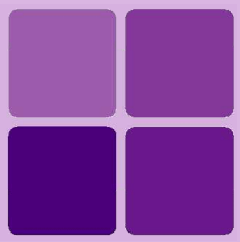


Intellicus Developers' Guide

Intellicus Enterprise Reporting and BI Platform

intellicus
Enterprise Reporting and
Business Insights Platform

info@intellicus.com
www.intellicus.com

Dated: April 2014

Acknowledgements

Intellicus acknowledges using of third-party libraries to extend support to the functionalities that they provide.

For details, visit: http://www.intellicus.com/acknowledgements.htm

Contents

Intellicus Integration Architectural Options

Intellicus Web Client supports all the Web servers and extensions that support JSP execution.

The Intellicus Web Client files include Intellicus Java Server Pages and Intellicus Java API packages. Other supporting files include java script files, image files, property configuration files, etc.

These files are available at <Intellicus_Install_path>\Jakarta\webapps\intellicus.

Integration Architectural Options

There are two recommended methods for integrating Intellicus client into a host web application.

- Intellicus portal can be run separately and can be interacted with HTTP calls, OR
- Intellicus portal JSPs and JAVA APIs can be copied inside the HOST application server.

Note: The options are with integrating the client components. In either case, the Report Server will run as a separate standalone server process.

Option-1: Intellicus Running on Separate Webserver

In this method Intellicus web client components are deployed in a separate application server and managed separately.

The client components run in the memory space of a separate Jakarta TomCat JSP Server. During streaming of report output to the browser, the client components won't consume host applications' threads, temporary file space etc.

Still, this integration option can provide embedded reporting, by placing reporting output as an integral part of the host application page.

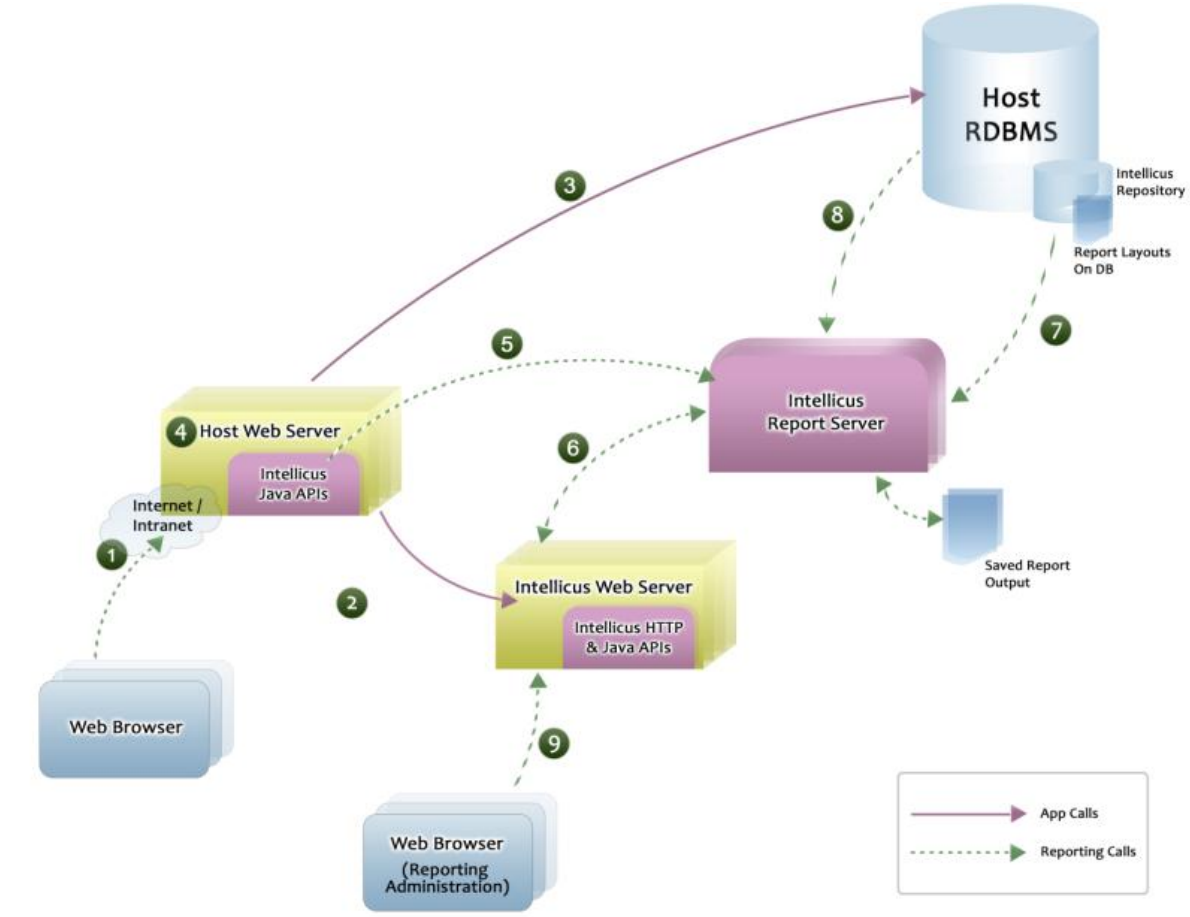


Figure 1: Integrating as a separate portal

Use cases (As numbered in the diagram)

1. Host application user login to their (Host) application.
2. User clicks Reports option for getting a report layout list etc. from the web browser and thus hit the Intellicus portal.
3. Host application user access Client database for their normal actions other than reports.
4. Client application communicates with Intellicus JAVA APIs (includes intellica.jar, xerces.jar, log4ij.jar, ReportClient.properties) deployed on Client app server in their class path.
5. Client uses Intellicus JAVA APIs for performing Intellicus administration task programmatically.
6. Intellicus portal communicates with report server for report execution, administration etc.
7. Intellicus report server communicates with Client database for fetching report layouts.(Intellicus creates its own repository tables in the database)
8. Intellicus report server communicates with Client database and fetches data using report SQL from Client database.

9. Host application user logs in through Intellicus portal for Admin Screens using Intellicus portal login screen.

Deploying JSPs and SERVLETs

Step 1- Install Intellicus Report Server

Run Setup.exe and Install Intellicus Report Server.

You can install Intellicus Report Server on a separate machine or on the same machine as the host application server is running.

The setup.exe installs the report server as well as the Intellicus client components.

Note: By default, Intellicus suite listens to HTTP requests on port 80. If this number conflicts with any of host application port numbers, then either of ports should be changed.

Step 2- Configure HTTP port number

Report Web Service will start automatically on port 80. To change Intellicus portal HTTP port number,

1. Go to [Intellicus_Install_path]\Program File \Intellicus\Jakarta\conf
2. Using any text editor, edit 'server.xml' file.
3. Search for Connector port="80" text and replace 80 with other number, for example, 8000.

The Intellicus can be now accessed as localhost:8000/intellicus.

Step 3- Setting Intellicus User Context

When it comes to integrating Intellicus with a host application it is desired that the user gets seamless experience. That means, user should not only get the same look and feel, but if the application needs user authentication, he/she should not be asked to log into Intellicus again.

Single sign-on refers to one time authentication performed by the host application. Intellicus does not perform any authentication check for the Users accessing Intellicus from Host Application as these are already authenticated. This means that User can access Intellicus without going through Intellicus Login page

Please refer IntellicusSingleSign-on.doc for more details on single sign on.

Note: IntellicusSingleSign-On.doc will be provided with Intellicus setup. Path: <Intellicus_Install_Path>\Docs\Manuals

Option-2: Intellicus Running inside a Host Application

In this method Intellicus web client components are deployed inside the host application server and managed along with the host application.

During streaming of report output to the browser, the client components run in the memory space of host application server and consume threads, temporary file space etc.

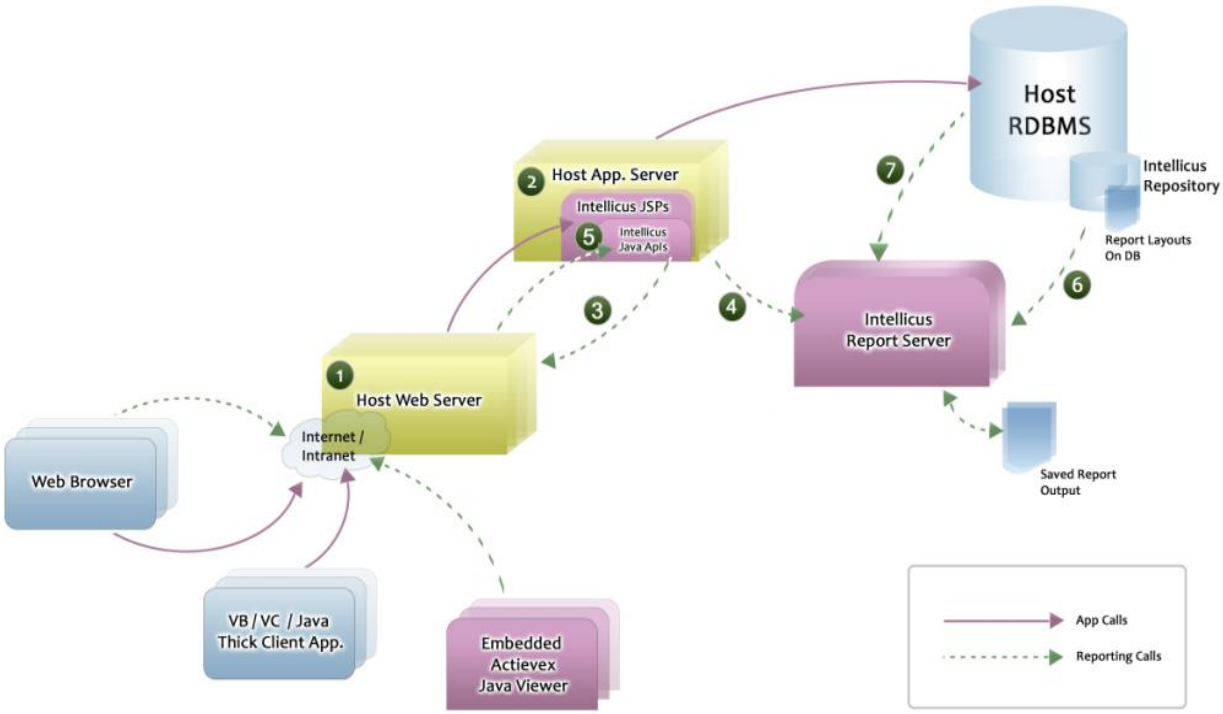


Figure 2: Integrated as embedded in host application

Use cases

1. Client application is a web-based application, which is deployed on some App Server. Intellicus Web Application is running on the same server.
2. Intellicus web Application is embedded with the Host web application.
3. The Host Web Application accesses the Intellicus pages.
4. The Host Web Application accesses the Intellicus Report Server through Intellicus Web Application.
5. The Host application uses Intellicus JAVA APIs for creating/mapping Intellicus users/roles etc (Point 6 in the diagram). Host

application communicates with Intellicus JAVA APIs (includes Intellica.jar, Xerces.jar, log4ij.jar, ReportClient.properties) deployed on their App server in their class path.

6. Intellicus Report Server connects to Client database; with a single-user connection pool (Intellicus creates its own repository tables in the database). Intellicus repository is also formed on the same database (may be under a different schema).
7. Intellicus report server communicates with Client database and fetches data using report SQL from Client database.

Steps to deploy Intellicus as a Embedded Application inside Host Application

1. Copy "intellicus" web application to the Host Application

Go to the drive where Intellicus is installed and then follow the path Intellicus>Jakarta>webapps, copy the folder "intellicus" from there and place it inside the Host Application folder "hostapp".

```
<jakarta home>webapps\hostapp\
```

2. Copy library files of intellicus

Copy the intellica.jar, log4ij.jar, xercesImpl.jar and xmlParserAPIs.jar inside the lib folder of WEB-INF folder. The files intellica.jar, log4ij.jar, xercesImpl.jar and xmlParserAPIs.jar are taken from the lib folder of WEB -INF folder of the "intellicus" folder.

3. Modify the contents of "web.xml'

3.1 The contents of Intellicus' web.xml (inside the WEB-INF of intellicus folder) should be merged into web.xml of hostapp folder

```
<jakarta home>webapps\hostapp \WEB-INF\web.xml
```

3.2 Modify the entry of propertyFilename to set env-entry-value to /intellicus/client/config/ReportClient.properties

3.3 Also, the mappings mentioned below requires prefix as Application Folder Name in its uel-pattern. (Eg, if embedded folder "intellicus" is renamed as "report" inside host application, then "/report" should be used as the prefix)

```
<servlet-mapping>
      <servlet-name>ChartTemplateController</servlet-name>
      <url-pattern>/report/tools/amchart/templates/*</url-
pattern>
```

```
  </servlet-mapping>

  <servlet-mapping>
        <servlet-name>GridTemplateController</servlet-name>
        <url-pattern>/report/templates/grids/*</url-pattern>
  </servlet-mapping>

  <servlet-mapping>
        <servlet-name>HelpController</servlet-name>
        <url-pattern>/report/common/help/images/Default/*</url-
pattern>
  </servlet-mapping>

  <servlet-mapping>
        <servlet-name>HelpController</servlet-name>
        <url-pattern>/report/common/css/images/Default/*</url-
pattern>
  </servlet-mapping>

  <servlet-mapping>
         <servlet-name>HelpController</servlet-name>
        <url-
pattern>/report/common/help/stylesheets/help.css</url-pattern>
  </servlet-mapping>

  <servlet-mapping>
         <servlet-name>WebStudioUtility</servlet-name>
         <url-pattern>/report/webdesigner/images/user/*</url-
pattern>
   </servlet-mapping>

<env-entry>
    <env-entry-name>propertyFileName</env-entry-name>
    <env-entry-
value>/report/client/config/ReportClient.properties</env-entry-
value>
    <env-entry-type>java.lang.String</env-entry-type>
</env-entry>
```

4. Modify the ReportClient.properties file

The location of ReportClient.properties file

```
 <jakarta home>webapps\hostapp\intellicus\client\config
```

4.1 REPORT_ENGINE_IP section in ReportClient.properties file should point to Report Engine Server.

E.g.:

```
## [REPORT ENGINE CONFIGURATION]
# Address of machine where Report Engine is running
# (Default value:- 127.0.0.1)
REPORT_ENGINE_IP=192.168.33.110

# Port number on which Report Engine is running
# (Default Value:- 45450)
REPORT_ENGINE_PORT=45450
```

4.2 Set INTERA_HOME to blank.

4.3 Set RELATIVE_PATH=/intellicus

In case the name of the intellicus folder is renamed by the user like, "intellicus" is renamed as "report", then the relative path is set as "/report"

5. Modify the contents of "ReportEngine.properties"

The location of ReportEngine.properties:

```
<Intellicus_Install_Path>/ReportEngine/Config/
```

Change HYPERLINK_RELATIVE_PATH property

From : ../../../InteraController.jsp
To : ../../../**intellicus**/InteraController.jsp

In case the name of the intellicus folder is renamed by the user like, "intellicus" is renamed as "report", then HYPERLINK_RELATIVE_PATH property should be set as ../../../**report**/InteraController.jsp

6. Access Intellicus from hostapp application

User can access Intellicus with following URL

```
http://<IP Address> :< port no>/hostapp/intellicus/index.jsp
```

In case the name of the intellicus folder is renamed by the user like "intell icus" is renamed as "report" then the user can access Intellicus with the following URL:

```
http://<IP Address> :< port no>/hostapp/report/index.jsp
```

HTML Look and Feel

All Intellicus JSP files use the style sheet- 'intellicus.css' to generate the HTML look-and-feel. You can change 'intellicus.css', or modify these JSP files to use your cascading style sheet file (CSS) to give the look-and-feel according to your application.

Note: Intellicus provides professional services for developing the UI skin. Please contact Intellics support for the same.

HTTP API

When Intellicus and Host application are running on different web servers as described in deployment scenario 1, then Single Sign-On need to be implemented.

Please refer IntellicusSingleSign-on.doc for more details on single sign on.

Note: IntellicusSingleSign-On.doc will be provided with Intellicus setup. Path: <Intellicus_Install_path>\Docs\Manuals

For deployment scenario 2, in which Intellicus is embedded inside Host application, Single Sign-On is not required.

Intellicus functionalities can be accessed from within an application. This is done through HTTP APIs. APIs are available for following functionalities:

- Getting category and report list
- Run, Schedule, Publish, E-Mail report
- Dashboard, Widget listing, Setting/Getting Dashboard Preferences
- Query/Parameter Object Editor
- Adhoc reporting

General steps of integration are:

- Take Intellicus Token by implementing single sign-on
- Call HTTP API for the needed functionality
- Receive response from Intellicus
- Display the information in host application

Prerequisite

If the host application's users are expected to authenticate in order to enter into the application, host application will need to integrate with Intellicus using single sign-on (user authentication by host application).

Intellicus needs to make sure that a request received from a host application is from a user already authenticated by the host application. Intellicus provides token exchange mechanism for user identification.

Before calling Intellicus HTTP APIs, we assume that token had been generated by Intellicus and received by host application with the help of single sign on.

Method to get a token for Single Sign-On

This method calls Intellicus Controller API and passes the user credentials and other hidden/business/request parameters to Intellicus.

Name: getIntellicusToken
Class: SingleSignOn

Syntax:

public String getIntellicusToken() throws SingleSignOnException

Returns:

TokenString: Received token from Intellicus

Method to access HTTP API

This method is called after getting a token from Intellicus. This method sets the name of Intellicus HTTP API where request is being redirected.

Name: redirectToIntellicus
Class: SingleSignOn class

Syntax:

```
public void redirectToIntellicus(String onSuccess, String intellicusToken, HttpServletResponse response) throws IOException
```

Parameters:

- **onSuccess:** Relative URL of the requested Intellicus HTTP API.
- **intellicusToken:** Token received from Intellicus after user authentication.
- **response:** It is the HttpServletResponse object. It is used for redirecting a request.

CategoryList.jsp

The objective of the API is to fetch the all the categories.

REQ_CATEGORY_ID
(Optional)

This parameter is used to show only specified category.

Example: REQ_CATEGORY_ID= 4F9245A7-D639-4F99-604D-F32641B77725

REPORT_TYPE
(Optional)

This parameter is used to list the given type of reports in the specified category. Possible values for this parameter are STANDARD/STUDIO, ADHOC, OLAP and All. In case this parameter is not specified both types of reports will be listed.

Example: REPORT_TYPE= STUDIO.

Combination of the above parameters makes a relative URL that invokes category listing of Intellicus. For example:

```
String onSuccess
=/core/CategoryList.jsp?REPORT_TYPE=STUDIO&REQ_CATEGORY_ID=4F9
245A7-D639-4F99-604D-F32641B77725
```

ReportListForCategory.jsp

The objective of the API is to fetch the reports for a particular category. It also acts as an interface to perform any operations on the reports. This API is configurable on the basis of access rights and license.

With this API, user having the required access rights can perform all the report operations like run a report in desired format, schedule it, view saved reports and its description.

This API governs following behaviors:

- If the user is Super Admin or Admin he can perform all the activities.
- If the user is an end-user, and no access rights are granted, no icons will be visible.
- To quick run or run the report, 'run report' and 'publish' access rights are needed.
- To publish a report, 'view saved reports' right is needed.

- For scheduling, 'run report' and 'schedule report' rights are required.
- For editing Adhoc report, 'publish layout' rights are needed.

CATEGORY_ID
(Mandatory)

This is the id of the category in which the report being run exists. This API will list all the reports belonging to the specified category.

Example: CATEGORY_ID=4F9245A7-D639-4F99-604D-F32641B77725

CAT_NAME
(Optional)

This is the Category Menu Name to which the Report belongs.

Example: CAT_NAME=DemoCategory

REPORT_TYPE
(Optional)

This parameter is used when user wants to list which type of reports available in a specified category. Possible values for this parameter are **STANDARD/STUDIO**, **ADHOC**, **OLAP** and **ALL.**

In case this parameter is not specified, both types of reports will be listed.

Example: REPORT_TYPE=STANDARD

SEARCH_STRING
(Optional)

This parameter is used to display a list of reports having report name that is given in the SEARCH_STRING parameter.

Example: SEARCH_STRING=Country Sales

This API will list of all reports which contains 'Country Sales' in report name

FROM_DATE
(Optional)

This parameter is used for giving list of all reports which is updated from specified date.

Example: FROM_DATE = 06/06/2007

This API will list of all reports which updated from 06/06/2007 date.

TO_DATE
(Optional)

This parameter is used for giving list of all reports which is updated up to specified date.

Example: TO_DATE = 06/06/2008

This API will list of all reports which updated up to 06/06/2008 date.

If FROM_DATE and TO_DATE both parameters are given in HTTP API then it will fetch list of all reports which updated between these two dates

SHOW_RERUN
(Optional)

This parameter is used to enable/disable Re-Run option in the report listing page.

Example: SHOW_RERUN=true

This API will enable Re-Run Option in report listing.

SHOW_DELETE
(Optional)

This parameter is used to enable/disable delete option in the report listing page.

Example: SHOW_DELETE=true

This API will enable delete Option in report listing.

Combination of above parameters constructs a URL to get report listing of Intellicus reports. For example:

```
String onSuccess
=”/core/ReportListForCategory.jsp?REPORT_TYPE=ADHOC&CATEGORY_I
D=4F9245A7-D639-4F99-604D-
```

```
F32641B77725&SHOW_DELETE=true&FROM_DATE=06/06/2007&TO_DATE=06/
06/2009&SHOW_RERUN=true"
```

Sample URLs:

For getting list of all the reports in the category

```
/core/ReportListForCategory.jsp?CATEGORYID=96EF065A-92DE-5F64-
E2AF-C4139396DD6B
```

For getting list of Standards reports in the category

```
/core/ReportListForCategory.jsp?CATEGORYID=96EF065A-92DE-5F64-
E2AF-C4139396DD6B&REPORT_TYPE=STUDIO
```

For getting list of Adhoc reports in the category

```
/core/ReportListForCategory.jsp?CATEGORYID=96EF065A-92DE-5F64-
E2AF-C4139396DD6B&REPORT_TYPE=ADHOC
```

RepositoryExplorer.jsp

The objective of this API is to show the repository explorer to the User. This API is configurable on the basis of access rights and license.

With this API, user having the required access rights can view explorer for different entities like Reports, Categories, Queries, Parameter objects, Dashboards, Favorites etc.

ENTITYTYPE
(Optional)

This parameter takes entity that is to be shown in the Explorer.

Example: ENTITYTYPE = REPORT

Valid Values:
ENTITYTYPE = REPORT
ENTITYTYPE = FAVORITES
ENTITYTYPE = QUERY
ENTITYTYPE = PARAMETER
ENTITYTYPE = DASHBOARD2
ENTITYTYPE = DASHBOARD_WIDGET

In case this parameter is not specified, it takes "Repository" as default and shows Repository Explorer.

EXPLORER_TITLE

(Optional)

This parameter takes name that is to be shown in the Explorer Title.

Example: EXLPORER_TITLE = Reports

Combination of above parameters constructs a URL to get Object Explorer for Query Objects. For example:

```
String onSuccess
=”/core/RepositoryExplorer.jsp?ENTITYTYPE=QUERY&EXPLORER_TITLE
=Queries
```

Now pass above onSuccess parameter to redirectToIntellicus method of SingleSignOn class.

InteraController.jsp

This API is the main controller of Intellicus report server. All the reports related requests to the report server are passed through this API. This controller is used for both Standard and Adhoc reports. Depending on the Action code given, this controller will ask report server to do the required task.

InteraController.jsp is used to:

- Quick run a report
- Execute and view a report in a specific output format
- Execute and deliver a report as email, publish or printout
- View published report
- Schedule a report

This jsp accepts system parameters and business parameters.

System Parameters

ACTION_CODE
(Mandatory)

This parameter is to specify the action that this API will initiate. A list of action-codes and actions is provided in the table given below.

Action Code	Description
000	Takes navigation to system parameter page. From here a user can select report delivery option, database connection, output format and other options before running the report.
001	If the report has user parameters, it takes navigation to Intellicus user parameters page. User can specify values for parameters from this screen and can execute the report. If report does not have any parameters, it executes report directly.
010	To Run Report with previous run report parameter values.
002	Executes report and navigates to report viewer screen.
003	Request to show all the system, report parameter in one jsp.
300	Request to show all the system, eport parameter in one jsp.
004	Open Scheduled Repots in viewer.
005	Open published report in viewer.
400	Submission of edit parameter form and request for updated report.
500	Save dialog submission and request to save report.
600	Delete report layout.
700	Delete saved report.
800	Generate dynamic report.
802	Dynamic report generation for drilldown reports.
900	Promptable information check and execution of dynamic adhoc report with no parameter.
901	Execution of adhoc report after taking promptable information from AdHocWizardRun.jsp.
902	Promptable information check and execution of saved adhoc report with no parameter.

Example: ACTION_CODE=002

REPORT_ID
(Mandatory)

This is the Id of the report being run. User can view report id from Intellicus portal's "Deploy Categories and Reports" page.

Example: REPORT_ID=96EF065A-92DE-5F64-E2AF-C4139396DD6B

DSGN_MODE
(Optional)

This parameter is to specify designer of the report. The way report is executed, also depends on its design mode.

Valid values are:

- **STUDIO:** Specify this value when report being run is designed in Intellicus Studio (these reports are also known as Standard Reports).
- **ADHOC:** Specify this value when report being run is designed in Adhoc designer (these reports are also known as Adhoc Reports).

Default value: STUDIO
Example: DSGN_MODE=STUDIO

MENU_NAME
(Optional)

This is the name of the report being run. Report name is used in the UI for identifying the report.

Example: MENU_NAME=Sales by Country

CATEGORY_ID
(Optional)

This is the id of the category in which the report being run exists. User can view category id from Intellicus portal's "Deploy Categories and Reports" page.

Example: CATEGORY_ID=4F9245A7-D639-4F99-604D-F32641B77725

REPORT_FORMAT
(Optional)

This is the output format in which report is to view. Possible values for this parameter are **rdf, pdf, htm, xls, txt, rtf, ppt, dhtm(iHTML), csv, doc, xml, rwt(Rawtext), dhtm2(SMART)**

Default value: htm
Example: REPORT_FORMAT=htm

REPORT_CONN_NAME
(Optional)

This parameter is to specify name of the database connection to be used for running the report. Specified connection must exist in report server.

Example: REPORT_CONN_NAME=ReportDB

OPERATION_TYPE
(Optional)

This parameter is to specify type of the operation requested. Possible values are VIEW, EMAIL, SAVE, PRINT, PRINT_AT_SERVER.

- **VIEW:** Generated report is sent to the report viewer for display.
- **EMAIL:** Generated report is E-mailed as an attachment or hyperlink.
- **SAVE:** Report output is saved as snapshot instead of being displayed.
- **PRINT:** Report is sent to the local printer instead of being displayed.
- **PRINT_AT_SERVER:** Report is sent to the printer on report server instead of being displayed.
- **PRINT_LOCALLY:** Report is sent to the local printer instead of being displayed.

Default value: VIEW
Example: OPERATION_TYPE=VIEW

HTM_MULTIPAGEOUTPUT
(Optional)

This parameter is used to specify whether report output should be in multiple pages or single page. Possible values are true and false.

- **True:** Displays the report using multiple HTML pages.
- **False:** Displays report using a single HTML page by merging all report pages into one.

This parameter is used when OPERATION_TYPE is VIEW and REPORT_FORMAT is HTM.

Default value: true
Example: HTM_MULTIPAGEOUTPUT=true

HTM_SHOWTOOLBAR
(Optional)

This parameter is used to specify whether toolbar in the html report output should be displayed or not. Possible values are 0,1 and 2 for SHOW_NEVER, SHOW_ALWAYS, and SHOW_WHEN_MULTIPAGE respectively.

- **SHOW_NEVER:** Never shows the toolbar in HTML viewer. The viewer cannot navigate to further pages if the report has many pages.
- **SHOW_ALWAYS:** Always shows the toolbar in HTML viewer.
- **SHOW_WHEN_MULTIPAGE:** The toolbar is shown only when the report generates more than one page. Otherwise, when the report generates only one page the toolbar is hidden.

This parameter is used when OPERATION_TYPE is *VIEW* and REPORT_FORMAT is *HTM*.

Default value: SHOW_ALWAYS
Example: HTM_SHOWTOOLBAR= 1

SHOW_NEVER=0
SHOW_ALWAYS=1
SHOW_WHEN_MULTIPAGE=2

IRLDATA
(Optional)

It is used to get the IRL xml from the request object.

ARLDATA
(Optional)

It is used to get the ARL xml from the request object.

REQUESTTYPE
(Optional)

It is used to specify the type of Request related to particular Report whether to Cancel the Running Report or not.

Example: REQUESTTYPE = CANCEL

EXECUTIONTYPE
(Optional)

It is used to specify the Execution Type whether Run, Run In Background, or Scheduled

Example:
EXECUTIONTYPE = : ALL
EXECUTIONTYPE = **DIRECT**: Run
EXECUTIONTYPE = **SCHD**: Scheduled Report
EXECUTIONTYPE = **ASYNC**: Run in Background

STATUS
(Optional)

It is used to specify the Status of Report.

Example:
STATUS = : All
STATUS = **UNDERPROCESS**: Running
STATUS = **COMPLETED**: Completed

FROMDATE
(Optional)

This parameter is used for giving list of all reports updated from specified date.

Example: FROMDATE = 06/06/2007

This API will list of all reports updated from 06/06/2007 date.

TODATE
(Optional)

This parameter is used for giving list of all reports updated up to specified date.

Example: TODATE = 06/06/2008

This API will list of all reports updated up to 06/06/2008 date.

If both parameters, FROMDATE and TODATE are given in HTTP API then it will fetch list of all reports updated between these two dates.

ORPHAN
(Optional)

This specifies whether Orphan option is selected in Look in Category i.e. FRO MCATEGORY or from ReportLayout i.e. FROMREPORT.

Example: ORPHAN = **FROMCATEGORY**

APP_PRO_STATUS
(Optional)

This specifies the approval process status value.

Example: APP_PRO_STATUS = **All**

IS_DIRECT_EXEC
(Optional)

This specifies whether the Report should directly Run or show Parameters.

Example: IS_DIRECT_EXEC = False i.e. Show Parameters and then run the Report.

Business Parameters
(Optional)

Apart from system parameters and user parameters, host application can pass business parameters to Intellicus. These parameters can be used for enforcing authorization. Also business parameters can be used in report SQL for filtering records.

These parameters are used to specify query parameter names (and their respective values) in the HTTP URL.

Suppose we have a report having certain parameters, based on which it will fetch some records. For example a report named "country details" fetches records on the basis of country name. Parameter name is "prmCountry".

So we will send prmCountry parameter and its value in the URL.

Example: prmCountry='Brazil'

The URL to run a report (in HTML) accepting prmCountry as a run time parameter, the URL would be:

```
String onSuccess
="/InteraController.jsp?
ACTION_CODE=002&OPERATION_TYPE=VIEW&DSGN_MODE=STUDIO&R
EPORT_ID=2030DDEA-841B-E360-3CA0-
5954AA945B92&REPORT_FORMAT=htm&prmCountry='Brazil'
"
```

Sample URLs:

Run-time System parameters page

```
/InteraController.jsp?ACTION_CODE=000&REPORT_ID=96EF06
5A-92DE-5F64-E2AF-C4139396DD6B
```

Input parameters page

```
/InteraController.jsp?ACTION_CODE=001&REPORT_ID=96EF06
5A-92DE-5F64-E2AF-C4139396DD6B
```

For running standard report

```
/InteraController.jsp?ACTION_CODE=002&OPERATION_TYPE=V
IEW&DSGN_MODE=STUDIO&REPORT_ID=2030DDEA-841B-E360-
3CA0-5954AA945B92&REPORT_FORMAT=htm
```

For running Adhoc report

```
/InteraController.jsp?ACTION_CODE=002&OPERATION_TYPE=V
IEW&DSGN_MODE=ADHOC&REPORT_ID=2030DDEA-841B-E360-3CA0-
5954AA945B92&REPORT_FORMAT=htm
```

For running Adhoc report in pdf format

```
/InteraController.jsp?ACTION_CODE=002&OPERATION_TYPE=V
IEW&DSGN_MODE=ADHOC&REPORT_ID=2030DDEA-841B-E360-3CA0-
5954AA945B92&REPORT_FORMAT=pdf
```

OlapViewer.jsp

The objective of API is to show the list of OLAP reports available in the Repository.

This API takes below parameter

LAYOUT_ID
(Optional)
This parameter is used for opening a saved layout. When this parameter is specified, layout saved with this id should be opened and viewed in specified web browser's view port. Any valid Cube Layout ID saved in repository

```
 String                    onSuccess                    =
/olap/OlapViewer.jsp?LAYOUT_ID=F46DB80D-C532-5069-
F7A7-A43D726CB663
```

VIEW_MODE
(Optional)
This parameter is used to specify the view in which user wants to open the Viewer.

Possible Values

- GRID: To open the Viewer in Grid view
- CHART: To open the Viewer in Chart view
- DUAL: To open the Viewer in Dual Panel view

```
 String                    onSuccess                    =
/olap/OlapViewer.jsp?LAYOUT ID=F46DB80D-C532-5069-
F7A7-A43D726CB663&VIEW_MODE=CHART
```

DATA_ACTIONS
(Optional)

This parameter is used to specify whether User should be allowed to change data on Grid and Chart.

Possible Values

- TRUE: Allow Data Refreshing.
- FALSE: No Data Refreshing.
-

Default Value

- TRUE

```
 String                    onSuccess                    =
/olap/OlapViewer.jsp?LAYOUT_ID=F46DB80D-C532-5069-
F7A7-A43D726CB663&DATA_ACTIONS=FALSE
```

EXPLORER
(Optional)
This parameter is used to specify the desired state of Explorer Panel in the viewer.

Possible Values

- SHOW: To show Explorer Panel in expanded form.
- HIDE: To hide Explorer Panel
- COLLAPSED: To show Explorer Panel in collapsed form.

Default Value

- SHOW

Exceptions

- If *DATA_ACTIONS* =FALSE, then its value is always forced to HIDE

DISPLAY_TITLE
(Optional)
This parameter is used to specify whether title for layout should be displayed or not.

Possible Values

- SHOW: To show the title
- HIDE: To hide the title

Default Value

- SHOW

LAYOUT_ACTIONS
(Optional)
This parameter is used to specify whether Save and Open buttons should be displayed or not.

Possible Values

- TRUE: To show Save and Open buttons (Default)
- FALSE: To hide Save and Open buttons

Default Value

- TRUE

CONN_LIST
(Optional)
This parameter is used to specify whether Connection List should be displayed or not and if displayed it should be enabled or not.

Possible Values

- SHOW: To show enabled Connection list.
- HIDE: To hide connection list.
- DISABLE: To disable connection list.

Default Value

- SHOW

CO_LIST
(Optional)
This parameter is used to specify whether Cube Object List should be displayed or not and if displayed it should be enabled or not.

Possible Values

- SHOW: To show enabled Cube Object list.
- HIDE: To hide Cube Object list.
- DISABLE: To disable Cube Object list.

Default Value

- SHOW

OLAP_CONN_NAME
(Optional)
This parameter is used to specify the name of connection which should be selected by default in connection listing if present.

CO_NAME
(Optional)
This parameter is used to specify the name of Cube Object which should be selected by default in Cube Object listing.

TOOLBAR
(Optional)
This parameter is used to specify whether Toolbar should be displayed or not.

Possible Values

- SHOW: To show the Toolbar.

- HIDE: To hide the Toolbar.

Default Value

- SHOW

TOOLBAR_OPTIONS
(Optional)
This parameter is used to specify the toolbar options to be displayed.

The value is a comma-separated list containing the options which should be displayed in the toolbar.

- GRIDVIEW
- CHARTVIEW
- DUALVIEW
- SELECTCHART
- SWAPAXES
- CLEAR
- ACTION
- EXPORT
- ALL

Default Value

- ALL

SLICER
(Optional)
This parameter is used to specify whether Slicer Bar should be displayed or not.
Possible Values

- SHOW: To show the Slicer.
- HIDE: To hide the Slicer.

Default Value

- SHOW

SavedReportList.jsp

The objective of API is to show the list of saved reports available for the given report. Each saved report provides us with following details:

- Saved Report Name
- Published By
- Generation Time
- Expiry Time

- Comments

In addition to the above details, following options are also available:

- Option to view each saved report in any of the supported report output format.
- Option to view the comments provided by user(s) on that saved report. If collaboration is disabled in the license then when user clicks on the comment icon a message would pop-up indicating the same.
- Option to delete any of the saved report-instance(s).
- The Administrator (Super-Admin and Admin) are also provided with the option of viewing privately saved reports of other users.

REPORT_ID
(Mandatory)

This is the Id of the report for which saved reports list is requested.

Example: REPORT_ID=96EF065A-92DE-5F64-E2AF-C4139396DD6B

ISPUBLIC
(Optional)

This is to specify whether the report is public or private. Possible values for this parameter are true and false. In case this parameter is not specified both types of reports will be listed.

Example: ISPUBLIC=true

Combination of above parameters makes a URL that invokes saved report listing of Intellicus.

MENU_NAME
(Optional)

This is the name of the report being run. Report name is used in the UI for identifying the report.

Example: MENU_NAME=Sales by Country

CATEGORY_ID
(Optional)

This is the id of the category in which the report being run exists. User can view category id from Intellicus portal's "Manage Folders and Reports" page.

Example: CATEGORY_ID=4F9245A7-D639-4F99-604D-F32641B77725

FROM_DATE
(Optional)

This parameter is used for giving list of all reports updated from specified date.
(FROM_DATE shuold be in MM/DD/YYYY format).

Example: FROM_DATE = 06/06/2007

This API will list of all reports which updated from 06/06/2007 date.

TO_DATE
(Optional)

This parameter is used for giving list of all reports updated up to specified date.
(TO_DATE shuold be in MM/DD/YYYY format).

Example: TO_DATE = 06/06/2008

This API will list of all reports updated up to 06/06/2008 date.

If FROM_DATE and TO_DATE both parameters are given in HTTP API then it will fetch list of all reports updated between these two dates.

DEPTH
(Optional)

This parameter is to specify level of searching i.e. whether COMPLETE or CURRENT_LEVEL.
COMPLETE = -1 (Searches in complete multilevel category hierarchy)
CURRENT_LEVEL = 0 (Searches in current Category of multilevel hierarchy)

ACCESSRIGHTS
(Optional)

This parameter is to specify Access Rights for the Report.

CATACCESSRights
(Optional)

This parameter is to specify Access Rights for the given Category.

CALLEDFROM
(Optional)

It specifies the page, from where it is called.

Example: CALLEDFROM = VIEWER

ALL_USER_RPTS
(Optional)

This is the Request Parameter that is used to decide whether to get all the user's published report or not.

For example: **ALL_USER_RPTS** = true

```
String onSuccess = /core/ SavedReportList.jsp?
REPORT_ID=96EF065A-92DE-5F64-E2AF-C4139396DD6B&
ISPUBLIC=true&DEPTH=-1
```

Now pass above onSuccess parameter to redirectToIntellicus method of SingleSignOn class.

Dashboard

Objective of API: To load any dashboard by default or for editing any existing one. There are three types of APIs for dashboard:

- Dashboard Viewer API
- Widget Designer API
- Dashboard Preferences

DashboardViewer.jsp

This API takes below parameter.

DASHBOARD_ID: The unique identifier of the Dashboard that should be loaded for editing in the Dashboard viewer.

This parameter makes a URL that invokes Dashboard Viewer page of Intellicus.

For example:

```
String onSuccess =
/dashboard/DashBoardViewer.jsp?DASHBOARD_ID=DE65F377-
5E46-153B-A56E-54E1073D59B2
```

SELECTED_DASHBOARD_ID: This parameter shall be respected only if the dashboard with this id is in user's preferences.

For example:

```
String onSuccess =
/dashboard/DashBoardViewer.jsp?SELECTED_DASHBOARD_ID=B
954111B-1881-21CE-1958-0DB411187785
```

SHOW_FULL_SCREEN: This option is to show full screen option or not. If the value is false then it will not show the full screen option.

This parameter makes a URL that invokes Dashboard Viewer page with/without SHOW_FULL_SCREEN icon

For example:

```
String onSuccess =
/dashboard/DashBoardViewer.jsp?DASHBOARD_ID=DE65F377-
5E46-153B-A56E-54E1073D59B2&SHOW_FULL_SCREEN=true
```

WidgetDesigner.jsp

This API takes following parameters:

WIDGET_ID: The unique identifier of the Widget that should be loaded for editing.

This parameter makes a URL that invokes Widget Designer page of Intellicus.

SHOW_TAB (Optional)**:** This parameter is used to display Report Tabs on Widget Designer.

Possible Values: ALL/REALTIME/PREGEN
Default Value: ALL

DEFAULT_TAB (Optional)**:** This parameter is used to select default Report tab (Real Time or Pre-generated) on loading of Widget Designer.

Possible Values: REALTIME/PREGEN
Default Value: REALTIME

For example:

```
String onSuccess =
/dashboard/WidgetDesigner.jsp?WIDGET_ID=12628459959568
192168336132613160&SHOW_TAB=ALL&DEFAULT_TAB=PREGEN
```

DashboardPreferences.jsp

Below URL invokes the Dashboard Preferences page of Intellicus.

For example:

```
String onSuccess =/dashboard/DashBoardPreferences.jsp
```

AdhocWizard.jsp

Objective of the API: To load the Ad Hoc report designer and facilitate the user to perform all the desired activities on that UI. Moreover it is used to design new report as well as open and edit a saved report.

Parameters

- **DEFAULTEXPANDTABS:** All or None.
- **SYS_TEMPLATE_NAME:** is send as request to show template. It needs no validations and no pre defined values.
- **CATEGORY_ID:** Saves report in predefined category.(Only when designing a new Report and saving it)
- **REPORT_ID:** Opens the Report with given Report_Id for Editing.
- **LINK_TO_IDENTIFIER:** Provides tab identifier for launching use case on click of Query Editor.
- **REPORT_FORMAT**: This request parameter is used for governing default value for report format
- **WIZARD_LAYOUT_TYPE**: This request level parameter is used for defining layout type of Adhoc Wizard.

  Possible Values: TABBED/ACCORDION
- **SHOW_TAB*:*** HTTP request level parameter for selecting particular tab or section of Adhoc Wizard. This could be comma separated values.

  Possible Values: SELECT/GROUP/FILTER/TOTAL/SORT/HIGHLIGHT/ MATRIX/CHART/NETWORK_GRAPH
- **SHOW_CHART_TYPE*:*** HTTP request level parameter for showing specified chart type as default selected.

  Possible Values: PIE/BAR/LINE/CURVE/CURVEAREA/AREA/SCATTER/RADAR/BUBBLE

There are no mandatory parameters required in this case.

Combination of above parameters makes a URL that invokes AdhocWizard report design page of Intellicus.

For example:

Creating a new report

```
String onSuccess=/custom/AdHocWizard.jsp?
DEFAULTEXPANDTABS=All
```

Editing an existing report

```
String onSuccess=/custom/
AdHocWizard.jsp?DEFAULTEXPANDTABS=All&CATEGORY_ID=4F92
45A7-D639-4F99-604D-F32641B77725
```

Now pass above onSuccess parameter to redirectToIntellicus method of SingleSignOn class.

SelectAdhocSource.jsp

Objective of the API: To load the Ad Hoc report visualizer with Data source selection selection. This page is called when we open Adhoc visualizer from Navigation.

Adhoc Visualizer with Data Source selection Screen

```
String onSuccess=/custom/SelectAdhocSource.jsp
```

Now pass above onSuccess parameter to redirectToIntellicus method of SingleSignOn class.

AdhocVisualizer.jsp

Objective of the API: To load the Ad Hoc report visualizer and facilitate the user to perform all the desired activities on that UI. Adhoc Visualizer allows user to directly view the report data with all columns as default and then the user has provision to change certain things like selected columns, filters, grid view, chart view or Map view, apply grouping, sorting, filters, highlighting, total etc. Also the user can select existing report and view from this visualizer.

Parameters

VIEW_USECASE:

(Optional)

This parameter takes value of VIEW_USECASE parameter to allow user to open visualizer for various usecases like NEW for opening new report with given QUERY_ID or SAVED for opening existing report for given REPORTID etc.

Possible Values

- NEW- Used to design a new Report.QUERY_ID is mandatory. (Default)
- SAVED- Open Existing Report(REPORTID is mandatory)
- PUBLISHED- Open Saved instance of Report(REPORTOID is mandatory)
- REQUEST- To Open Adhoc Visualizer using ARLDATA.
- RUNINBACKGROUND- Run Smart Report in background(REPORTID is mandatory)
- AUDIT- Called for Auditted Reports (REPORTOID is mandatory)
- CHANGEDATASOURCE- To Open Report with different Data Source

Adhoc Visualizer with Data Source loaded for designing New Report

```
String onSuccess=/custom/AdhocVisualizer.jsp?
VIEW_USECASE=NEW&QUERY_ID=126077293450092031291929106718 82
```

Adhoc Visualizer opened for Existing Report

```
String onSuccess=/custom/AdhocVisualizer.jsp?
VIEW_USECASE=SAVED&REPORT_ID=AF654266-1477-EE8A-BFB1-
5D0A2A59A4BE
```

QUERY_ID:

This parameter is mandatory for VIEW_USECASE as NEW. It takes Query Id which is to be loaded in Adhoc Visualizer.

Open Adhoc Visualizer with Data Source loaded

```
http://localhost:91/intellicus/custom/AdhocVisualizer.jsp?VIE
W_USECASE=NEW&QUERY_ID=126077293450092031291929106718 82
```

REPORTID:

This parameter is mandatory for VIEW_USECASE as SAVED. It takes Report Id so as to load the existing Report in Adhoc Visualizer.

Open Adhoc Visualizer with Report loaded

```
http://localhost:91/intellicus/custom/AdhocVisualizer.jsp?VIEW_USECASE=SAVED&REPORT_ID=AF654266-1477-EE8A-BFB1-5D0A2A59A4BE
```

EDIT_MODE:

(Optional)

This parameter takes value of **EDIT_MODE** parameter to allow user to open visualizer in Edit mode or View Mode.

Possible Values

- TRUE: Opened with Edit Mode
- FALSE: Opened in View Mode.

Adhoc Visualizer with Data Source loaded in view mode

```
String onSuccess=/custom/AdhocVisualizer.jsp?QUERY_ID=126077293450092031291929106718 82&EDIT_MODE=FALSE
```

Adhoc Visualizer with Data Source loaded in Edit mode

```
String onSuccess=/custom/AdhocVisualizer.jsp?QUERY_ID=126077293450092031291929106718 82&EDIT_MODE=TRUE
```

Combination of above parameters makes a URL that invokes AdhocWizard report design page of Intellicus.

Now pass above onSuccess parameter to redirectToIntellicus method of SingleSignOn class.

QueryObjectList.jsp

To display the detail of a query object.

Parameters

QUERY_ ID
(Optional)

To specify the query object's Id to view details.

```
onSuccess =
/custom/reportobjects/QueryObjectList.jsp?QUERY_ID=12607732892
6986838754445
```

QUERY_NAME
(Optional)

To specify the query object's name to view its details.

```
onSuccess =
/custom/reportobjects/QueryObjectList.jsp?QUERY_NAME=Sales
Detail
```

ParameterObjectList.jsp

To display the detail of a query object.

Parameters

PARAMETER_NAME
(Optional)

To specify the Parameter object's name to view its details.

```
onSuccess =
/custom/reportobjects/ParameterObjectList.jsp?PARAMETER_NAME=p
rmCountry
```

PARAMETER_ID
(Optional)

To specify the Parameter Id of the parameter to view its details.

```
/custom/reportobjects/ParameterObjectList.jsp?PARAMETER ID=126
467746605051270180911946549 91
```

PrintSettingList.jsp

This API is used to call page that have functionalities of working with print related settings. When a report is associated with a print setting, report is printed as per the preferences set in the associated print setting.

For example:

```
onSuccess =
/core/PrintSettingsList.jsp?PrintSettingsName=PrintSetting1
```

Preferences.jsp

This API provides UI to set following user preferences:

- Change password
- Set default connection
- Set portal preferences
- Set parameter preferences
- Set dashboard preferences
- Messenger preferences

For example:

```
onSuccess = personalization/Preferences.jsp
```

RESTful API

Intellicus provides RESTful APIs for Report Execution in the desired format. These APIs are developed using Jersey implementation. Using these Intellicus Rest APIs Report output can be saved in desired format.

Configuration

Download Jersey libraries

Intellicus RESTful APIs use following Jersey libraries:

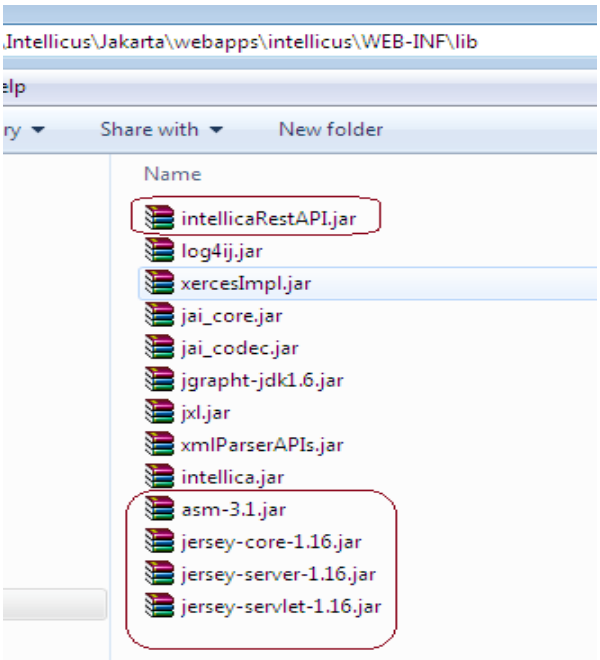
- asm.jar
- jersey-core.jar
- jersey-server.jar
- jersey-servlet.jar

These jars are not shipped by Intellicus. You need to download the zip of Jersey contains all the the Jersey jars.

Place the above jars in Intellicus web application at location:

<Intellicus Install path>\jakarta\webapps\intellicus\WEB-INF\lib

Make sure that you have intellicaRestAPI.jar in web application.



Servlet mappings in web.xml

Add the below entries in web.xml file-
(<Install_Path>\jakarta\webapps\intellicus\WEB-INF\web.xml)

```
<display-name>Rest API</display-name>
```

```
  <servlet>
       <servlet-name>Jersey REST Service</servlet-name>
       <servlet-class>
com.sun.jersey.spi.container.servlet.ServletContainer</servlet-
class>
      <init-param>
        <param-name>
com.sun.jersey.config.property.packages</param-name>
        <param-value>com.intellicus.API.RestAPI</param-value>
      </init-param>
      <load-on-startup>1</load-on-startup>
   </servlet>
   <servlet-mapping>
      <servlet-name>Jersey REST Service</servlet-name>
      <url-pattern>/rest/*</url-pattern>
    </servlet-mapping>

</web-app>
```

Note: Restart the Web-server after making the configuration changes.

Call RESTful API

Report Execution

Intellicus Report Execution Rest API is available for both GET and POST type requests.
For user authentication, GET implementation takes Intellicus Token as query parameter. In POST requests Intellicus Token or user credentials can be passed as request parameters.

1. GET Request

```
http://
<Intellicus_Server_IP>:<PORT>/intellicus/rest/ReportExec/query?
REPORT_ID=E00B3E24-9034-D022-134F-
53B19687C8A4&REPORT_FORMAT=xls&SYS_ZIPPED=FALSE&INTELLICUS_TOKEN=
"+intellicusToken
```

2. POST Request

```
http://<Intellicus_Server_IP>:<PORT>/intellicus/rest/ReportExec
```

Other parameters like REPORT_ID, REPORT_FORMAT, INTELLICUS_TOKEN or User credentials- USER_CREDENTIAL_1(UserId), USER_CREDENTIAL_2(Password) and USER_CREDENTIAL_3(OrganizationId) can be send in request using POST method.

Note: Intellicus Token is used by intellicus for authentication using Single Sign On. Please refer IntellicusSingleSign-on.doc for SSO implementation.

Please refer to Appendix-2 for sample codes to call REST API using POST and GET.

Java API

Intellicus Java APIs can be used for performing activities like User management, and Report Management in Intellicus.

To use the Java APIs of Intellicus, following jars should be placed in class path of the web application.

- intellica.jar
- log4ij.jar
- xercesImpl.jar
- xmlParserAPIs.jar

These jars are provided with intellicus setup. Following is given the path for jar file:

<Intellicus Install path>\APIs

Java Doc

The Java Doc for the APIs is available at FTP locaton: ftp://demo:demo@ftp.intellicus.com/api/ .

Mandatory Step to Use Java APIs

Initialize Report Client

Configuration

Intellicus client JAVA APIs are configured by a file "ReportClient.properties".

This file must be present in the JAVA CLASS_PATH variable of the host java application.

During initialization action, the report client component reads the configuration file and keeps the configuration in memory.

Import

Host java application class has to import the following classes, to initialize the Report Client.

```
import com.intellica.client.init.ReportClient
import java.io.FileInputStream
import java.io.BufferedInputStream
```

```
import java.io.InputStream
```

The initialization action is to be performed once in the lifetime of application.The initialization would be done according to the configurations set in the ReportClient.properties file

Init

Init method is used to initialize the Intellica client SDK with the values from the ReportClient.properties file. This properties file contains configurations for Report Engine Interface.

```
InputStream is= new BufferedInputStream (new FileInputStream
("<ReportClient.properties_AbsolutePath>"));
//Static method.
ReportClient.init (is);
```

If it is required to create the logs at desired location, set the absolute or relative path in the INTERA_HOME property of the ReportClient.Properties file.

Eg- If ReportClient.properties is placed at location c://client/config/, (i.e. client complete folder is copied down in c:// drive),

```
InputStream is= new BufferedInputStream(new
FileInputStream("C://client/config/ReportClient.properties"));
com.intellica.client.init.ReportClient.init(is);
```

then INTERA_HOME property should be set as –

Absolute Path-

```
//setting absolute path for logs
INTERA_HOME=C://client
```

Relative Path-

```
//setting relative path for logs
INTERA_HOME=../../../client
```

(This path is basically relative to jakarta/bin/, therefore, if we consider that intellicus is installed at loc- c://program files/intellicus/jakarta/bin, then relative path for logs(i.e. C://client/logs/) with respect to bin would be ../../../client)

Initialize Requestor User context

Requestor User

A Requestor User is the user, who is requesting any action to the Intellicus system.

A com.intellica.client.common.UserInfo class object represents a user in Intellicus.

The UserInfo class has the following attributes.

Attribute	Description
userId	User ID
password	Password
orgID	Organization ID
sessionId	Session ID
securityDescriptor	Security Descriptor string
customerId	Customer ID for service provide deployments
location	Location ID
locale	Locale
dbName	Database Name

Intellicus mode of authentication uses **User ID**, **Password** and **Org ID** as mandatory attributes to authenticate a user and authorize that user's actions.

A host application that takes over authentication responsibilities can use any of the above attributes to authenticate.

Import

Java Application class has to import the following class to store the login user information and check for Authorization.

```
import com.intellica.client.common.UserInfo;
```

UserInfo

The UserInfo object acts as a carrier to all above attributes in such case.

So, in all the use cases discussed below, host application has to create an object of UserInfo class and pass it in all the method calls.

```
UserInfo requestorUserInfo = new
        UserInfo("John","john","Org1");
```

Note: For performing any admin activity like user management at Intellicus, requestor user should be admin user at Intellicus.

Use Cases

User Management Actions

Import

```
//User Management imports
import com.intellica.client.security.SecurityManager;
import com.intellica.client.exception.ISecurityException;
```

```
import com.intellica.client.security.Authentication;

//Organization specific imports
import com.intellica.client.security.OrgInfo;

//User/Role specific imports
import com.intellica.client.security.RoleInfo;
import
com.intellica.client.common.Enums.SecurityTypes.ReportPrivileges;
import com.intellica.client.common.Enums;

import java.util.Vector;
import java.util.HashMap;
```

Organization

GetOrgById

This Java API is used to get an Organization for given Organization Id.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/Organization Management/ GetOrgById.java for sample code of this use case.

Steps :

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a SecurityManager class object for getting the controller information for all Administration related operations.

```
SecurityManager sMgr=SecurityManager.getInstance();
```

4. Get the Organization by OrgId and get its various details.

Method: getOrganizationById

```
public  getOrganizationById(java.lang.String orgId,
                                    userInfo)
                              throws
```

Method will return the organization Info for given orgId

Parameters:

- **orgId**: Id of the requested OrgInfo object

- **userInfo**: The current user details

Returns:
OrgInfo object having the given orgId (may return null)

```
//This will get the Information about any existing Organization
String organisationId="Test";
OrgInfo orgObj = sMgr.getOrganizationById( organisationId,
                                           requestorUserInfo);

System.out.println ("OrganizationName: "+orgObj.getOrgId());
System.out.println("Description: "+orgObj.getOrgDescription());
System.out.println ("OrganizationMapType: "+orgObj.getMapType());
System.out.println("Authorization mode: "
                   +orgObj.getAuthorizeMode());
System.out.println("AuthenticationObject:
                   "+orgObj.getAuthenticationObj());
```

Get Users list for Organization

This Java API is used to get list of users for the given organization in the Intellicus Repository. User is required to provide the organisation id, whose list of users, the requesting user wants to retrieve.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/Organization Management/GetUserListForOrg.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a SecurityManager class object for getting the controller information for all Administration related operations.

```
SecurityManager sMgr = SecurityManager.getInstance();
```

4. Get the Arraylist containing all users in the given Organization

Method: getUserListForOrganization

```
public                                          java.util.ArrayList
getUserListForOrganization(java.lang.String orgId,
 userInfo)
throws
```

This method returns the list of users for the given Organization ID.

Parameters:

OrgId: of the userInfo list to be retrieved.

Returns:

ArrayList of UserInfo Bean class

```
ArrayList arrList=new ArrayList();
String orgId="Intellica";
arrList = sMgr.getUserListForOrganization(orgId,
requestorUserInfo);
```

Get Organization List

This Java API is used to get the list of all organizations present in the Intellicus repository.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/Organization Management/ GetOrgList.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a SecurityManager class object for getting the controller information for all Administration related operations.

```
SecurityManager sMgr=SecurityManager.getInstance();
```

4. Get the Arraylist containing all users in the given Organization

Method: getOrgList

```
public java.util.ArrayList getOrgList( userInfo)
                                throws
```

This method returns the list of all organizations created in Intellicus repository.

Returns:

ArrayList of OrgInfo objects.

```
//An arraylist created to hold the list of organizations in it.
ArrayList arrList=new ArrayList();
```

```
//Method returns list of all organizations in Intellicus repository
arrList=sMgr.getOrgList(requestorUserInfo);
```

Add Organization

This Java API is used to add a new organization in the Intellicus Repository and set the authentication mode.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/Organization Management/**AddOrg.java** for sample code of this use case.

Also, you may refer to **AddCallbackOrg.java** for creating an organization whose authentication check is performed by Callback Mechanism.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create instance of OrgInfo.

```
String organisationId="Test";
OrgInfo addOrg=new OrgInfo(organisationId);
```

4. Set Authentication mode.

Method: setAuthenticateMode

public void setAuthenticateMode(int authenticateMode)

Authentication Mode Id values
1= Intellicus
(com.intellica.client.common.Enums.SecurityTypes.Authentication.*REPORTINGSYS)*

2= External Application
(com.intellica.client.common.Enums.SecurityTypes.Authentication.
EXTERNALAPP)

3= Host Application
(com.intellica.client.common.Enums.SecurityTypes.Authentication. *HOSTAPP)*

4= Callback
(com.intellica.client.common.Enums.SecurityTypes.Authentication. *CALLBACK)*

This identifies who will authenticate the users/roles belonging to this organization.

Parameters:
`authenticateMode` - : Authentication Mode Id's value

```
Authentication authInfo = new Authentication();
```

```
authInfo.setAuthenticateMode(com.intellica.client.common.Enums.SecurityTypes.Authentication.HOSTAPP);
addOrg.setAuthenticationObj(authInfo);
```

5. Create a SecurityManager class object for getting the controller information for all Administration related operations.

```
SecurityManager sMgr=SecurityManager.getInstance();
```

6. Add the Organization.

Method: addOrganization

```
public void addOrganization( orgInfo,
                             userInfo)
                      throws
```

This method adds a new Organization at the Report Engine.

Parameters:

- **OrgInfo:** The Organization details
- **UserInfo:** The User details

```
sMgr.addOrganization(addOrg, requestorUserInfo);
```

Modify Organization

This Java API is used to modify the Organization's details.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/Organization Management/ ModifyOrg.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Get the Organization i.e. to be deleted By its Organization Id.

```
String organisationId="Test";
OrgInfo modOrg=sMgr.getOrganizationById( organisationId,
requestorUserInfo);
```

4. Set the attributes of Organization

```
modOrg.setCanDelete(true);
String description="THIS IS A TEST ORGANIZATION";
```

```
modOrg.setOrgDescription(description);
```

5. Create a SecurityManager class object for getting the controller information for all Administration related operations.

```
SecurityManager sMgr=SecurityManager.getInstance();
```

6. Modify the Organization.

Method: modifyOrganization

```
public void modifyOrganization( orgInfo,
                                userInfo)
                          throws
```

This method modifies an Organization at the Report Engine

Parameters:

- **orgInfo**: The Organization details
- **userInfo**: The current user

```
sMgr.modifyOrganization(modOrg,requestorUserInfo);
```

Delete Organization

This Java API is used to delete an existing organization from the Intellicus repository.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/Organization Management/DeleteOrg.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Get the Organization i.e. to be deleted By Organization Id.

```
String organizationId="Test";
OrgInfo    delOrg    =    sMgr.getOrganizationById(organizationId,
requestorUserInfo);
```

4. Create a SecurityManager class object for getting the controller information for all Administration related operations.

```
SecurityManager sMgr=SecurityManager.getInstance();
```

5. Delete the Organization.

Method: deleteOrganization

```
public void deleteOrganization( orgInfo,
                                userInfo)
                          throws
```

This method deletes an Organization at the Report Engine from the Intellicus Repository.

Parameters:

- **orgInfo**: The Organization details
- **userInfo**: The current user details

```
sMgr.deleteOrganization(delOrg,requestorUserInfo);
```

Assign Category Privileges to Organization

This Java API is used to assign the Access privileges of a Category to the specific Organization.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/User Management/ AssignCategoryPrivilegesToOrg.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a SecurityManager class object for getting the controller information for all Administration related operations.

```
SecurityManager sMgr=SecurityManager.getInstance();
```

4. Create instance of AppInfo that contains the Organization to which the access rights are to be assigned.

```
  String userId="";
  String OrgId="TestOrg";
  //AppType for which the privileges are to be set, "" for Org,
"USER" for User and "ROLE" for Role
  String appType = "";

//Create instance of AppInfo that contains entity for which
access rights are to be assigned
AppInfo appInfo = new AppInfo(userId,OrgId,appType);
```

```
```

5. Create instance of EntityInfo which contains entity whose access rights are assigned on Category.

```
//Set desired entity : CATEGORY/REPORT/QO/PO/CUBEOBJECT
/DASHBOARD2/DASHBOARD_WIDGET/DBCONNECTION
EntityInfo entityInfo=new EntityInfo(catId,"CATEGORY");
```

6. Create object of EntityAccessRight and set accessrights

```
EntityAccessRight ear=new EntityAccessRight();
```

```
ear.setAppInfo(appInfo);
/*      access level can have below possible values-
FULLACCESS, PARTIALACCESS, DENYACCESS, NONEACCESS
  * public void setAccessLevel(int accessLevel)
  * @param accessLevel : access level can be 0/1/2/3
*/
ear.setAccessLevel(Enums.SecurityTypes.AccessLevel.PARTIALACCESS)
;
 ear.setAccessRightGrants("0,1,2");
```

7. Setting access rights for Reports, Dashboards under this category

```
//Access Rights for Reports under this category
EntityGroupAccessRight reportEAR=new EntityGroupAccessRight();
reportEAR.setType("REPORT");
reportEAR.setAccessLevel(1);//0 for deny, 1 for Full, 2 for
Partial
reportEAR.setGrants("10,12,14");
ear.addEntityGroupAccessRight(reportEAR);

//Access Rights for Dashboards under this category
EntityGroupAccessRight dashboardEAR=new EntityGroupAccessRight();
dashboardEAR.setType("DASHBOARD2");
dashboardEAR.setAccessLevel(1);//0 for deny, 1 for Full, 2 for
Partial
ear.addEntityGroupAccessRight(dashboardEAR);

ear.setOpCode("REPLACE");
```

Method: grantEntityPrivileges

This API allows the user to assign access rights information to a user/role/organization or Everyone on an entity.

Syntax

```
public void grantEntityPrivileges(EntityInfo entity,
EntityAccessRight entityAccessRight,
UserInfo userInfo)throws ISecurityException.
```

Parameters:

- **entity -** The entity object.This object must be created by setting entityId and entityType. entityTypes supported are defined in EntityTypeNames class..
- **entityAccessRight:** The entityAccessRight object. This object should contain the AppInfo object with the credentials of the user to which grants are to be assigned and access level as defined in Enums.SecurityTypes.AccessLevel
- **UserInfo:** Details of current user who is providing access rights.
- **Throws**: ISecurityException: - If the request cannot be performed successfully. This happens - 1.if connection can't be established with the engine or 2.if read or write operation cant be performed from or to the engine. 3.If some error has occured while executing the request on the engine. 4.If the response xml obtained from server cannot be parsed

Example

Given below is the example of actual implementation of this method:

```
sMgr.grantEntityPrivileges(entityInfo, ear, requestorUserInfo);
```

Assign Connection Privileges to Organization

This Java API is used to assign the Connection privileges to the specific Organization.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/User Management/ AssignConnectionPrivilegesToOrg.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a SecurityManager class object for getting the controller information for all Administration related operations.

```
SecurityManager sMgr=SecurityManager.getInstance();
```

4. Create instance of AppInfo that contains the UserId, Organization of the user to which the access rights are to be assigned.

```
  String appId="";
  String OrgId="TestOrg";
  //AppType for which the privileges are to be set, "" for Org,
"USER" for User and "ROLE" for Role
  String appType = "";

//Create instance of AppInfo that contains entity for which
access rights are to be assigned
AppInfo appInfo = new AppInfo(userId,OrgId,appType);
```

5. Create instance of EntityInfo which contains entity whose access rights are assigned on Category.

```
// Connection whose privileges are to be set for the specified
Organization
String entityId="DemoReportDB";//Category Id on which
AccessRights will be given to the user;

//Set desired entity :
CATEGORY/REPORT/QO/PO/CUBEOBJECT/DASHBOARD2/DASHBOARD_WIDGET/DBCO
NNECTION
EntityInfo entityInfo=new EntityInfo(entityId,"DBCONNECTION");
```

6. Create object of EntityAccessRight and set accessrights

```
EntityAccessRight ear=new EntityAccessRight();
```

```
ear.setAppInfo(appInfo);
/*      access level can have below values possible.
 *      Full Access
 *   Deny Access
**/
ear.setAccessLevel(Enums.SecurityTypes.AccessLevel.FULLACCESS);
ear.setOpCode("REPLACE");
```

Method: grantEntityPrivileges

This API allows the user to assign access rights information to a user/role/organization or Everyone on an entity.

Syntax

```
public void grantEntityPrivileges(EntityInfo entity,
EntityAccessRight entityAccessRight,
UserInfo userInfo)throws ISecurityException.
```

Parameters:

- **entity -** The entity object.This object must be created by setting entityId and entityType. entityTypes supported are defined in EntityTypeNames class..
- **entityAccessRight:** The entityAccessRight object. This object should contain the AppInfo object with the credentials of the user to which grants are to be assigned and access level as defined in Enums.SecurityTypes.AccessLevel
- **UserInfo:** Details of current user who is providing access rights.
- **Throws**: ISecurityException: - If the request cannot be performed successfully. This happens - 1.if connection can't be established with the engine or 2.if read or write operation cant be performed from or to the engine. 3.If some error has occured while executing the request on the engine. 4.If the response xml obtained from server cannot be parsed

Example

Given below is the example of actual implementation of this method:

```
sMgr.grantEntityPrivileges(entityInfo, ear, requestorUserInfo);
```

User

Get User By Id

This Java API is used to get User's information for the given userid.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/User Management/GetUserById.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a SecurityManager class object for getting the controller information for all Administration related operations.

```
SecurityManager sMgr=SecurityManager.getInstance();
```

4. Get the User by User Id.

Method: getUserById

```
public  getUserById(java.lang.String userId,
                            java.lang.String orgId,
                             userInfo)
                     throws
```

This method returns the UserInfo class object having the given userId of the given orgId

Parameters:

- **userId**: Id of the requested UserInfo object.
- **orgId**: Organization Id of the requested user.
- **userInfo**: The current user details.

Returns:

UserInfo: object having the given userId (may return null)

```
String userId = "Mary";
String orgId = "MyOrg";

UserInfo userInfo=sMgr.getUserById(userId, orgId ,
                                   requestorUserInfo);
```

Add User (Create User)

This Java API is used to create a new User in Intellicus Repository.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/User Management/ AddUser.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a SecurityManager class object for getting the controller information for all Administration related operations.

```
SecurityManager sMgr=SecurityManager.getInstance();
```

4. Create instance of new User and set its attributes.

```
String username = "Mary";
String password = "123456";
String orgId = "Org1";

UserInfo addUserInfo = new UserInfo(username, password, orgId);

// Optionally, make the user admin for that organization.
addUserInfo.setAdmin(true);
```

```
// Optionally, make the user Super admin.
addUserInfo.setSuperAdmin(true);
```

5. Add the User in Intellicus Repository.

Method: addUser

```
public void addUser( newUserInfo,
                     userInfo)
             throws
```

Adds a new User at the Report Engine.

Parameters:

- **newUserInfo**: The new user or target user details.
- **userInfo**: The current user details.

```
sMgr.addUser(addUserInfo,requestorUserInfo);
```

Assign Category Privileges to User

This Java API is used to assign the Access privileges of a Category to the specific Organization.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/User Management/ AssignCategoryPrivilegesToOrg.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a SecurityManager class object for getting the controller information for all Administration related operations.

```
SecurityManager sMgr=SecurityManager.getInstance();
```

4. Create instance of AppInfo that contains the UserId, Organization of the user to which the access rights are to be assigned.
   Also set AppType="USER"

```
  String userId="Mary";
  String OrgId="TestOrg";
  //AppType for which the privileges are to be set, "" for Org,
"USER" for User and "ROLE" for Role
  String appType = "USER";
```

```
//Create instance of AppInfo that contains entity for which
access rights are to be assigned
AppInfo appInfo = new AppInfo(userId,OrgId,appType);
```

5. Create instance of EntityInfo which contains entity whose access rights are assigned on Category.

```
//Set desired entity : CATEGORY/REPORT/QO/PO/CUBEOBJECT
/DASHBOARD2/DASHBOARD_WIDGET/DBCONNECTION
EntityInfo entityInfo=new EntityInfo(catId,"CATEGORY");
```

6. Create object of EntityAccessRight and set accessrights

```
EntityAccessRight ear=new EntityAccessRight();
```

```
ear.setAppInfo(appInfo);
/*      access level can have below possible values-
FULLACCESS, PARTIALACCESS, DENYACCESS, NONEACCESS
  * public void setAccessLevel(int accessLevel)
  * @param accessLevel : access level can be 0/1/2/3
*/
ear.setAccessLevel(Enums.SecurityTypes.AccessLevel.PARTIALACCESS)
;
 ear.setAccessRightGrants("0,1,2");
```

7. Setting access rights for Reports under this category

```
//Access Rights for Reports under this category
EntityGroupAccessRight reportEAR=new EntityGroupAccessRight();
reportEAR.setType("REPORT");
reportEAR.setAccessLevel(1);//0 for deny, 1 for Full, 2 for
Partial
reportEAR.setGrants("10,12,14");

ear.addEntityGroupAccessRight(reportEAR);

ear.setOpCode("REPLACE");
```

Method: grantEntityPrivileges

This API allows the user to assign access rights information to a user/role/organization or Everyone on an entity.

Syntax

```
public void grantEntityPrivileges(EntityInfo entity,
EntityAccessRight entityAccessRight,
UserInfo userInfo)throws ISecurityException.
```

Parameters:

- **entity -** The entity object.This object must be created by setting entityId and entityType. entityTypes supported are defined in EntityTypeNames class..
- **entityAccessRight:** The entityAccessRight object. This object should contain the AppInfo object with the credentials of the user to which grants are to be assigned and access level as defined in Enums.SecurityTypes.AccessLevel
- **UserInfo:** Details of current user who is providing access rights.
- **Throws**: ISecurityException: - If the request cannot be performed successfully. This happens - 1.if connection can't be established with the engine or 2.if read or write operation cant be performed from or to the engine. 3.If some error has occured while executing the request on the engine. 4.If the response xml obtained from server cannot be parsed

Example

Given below is the example of actual implementation of this method:

```
sMgr.grantEntityPrivileges(entityInfo, ear, requestorUserInfo);
```

Assign Report Privileges To User

This Java API is used to set the Report Access Privileges for a particular user. The user would require to provide its identity i.e. user Info and the Report for which the access rights are to be given.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/User Management/ AssignReportPrivilegesToUser.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a SecurityManager class object for getting the controller information for all Administration related operations.

```
SecurityManager sMgr=SecurityManager.getInstance();
```

4. Create instance of AppInfo that contains the UserId, Organization of the user to which the access rights on report are to be assigned.

Also set AppType="USER"
//Set appId = <ROLE_NAME> and appType="ROLE"

```
  String appId="Mary";
  String OrgId="TestOrg";
  //AppType for which the privileges are to be set, "" for Org,
"USER" for User and "ROLE" for Role
  String appType = "USER";

//Create instance of AppInfo that contains entity for which
access rights are to be assigned
AppInfo appInfo = new AppInfo(appId,OrgId,appType);
```

5. Create instance of EntityInfo which contains entityID/ReportID for which access rights are to be assigned

```
String reportId="D07131A2-87AA-154E-6E17-6079C9AFD176";
//Set desired entity : CATEGORY/REPORT/QO/PO/CUBEOBJECT
/DASHBOARD2/DASHBOARD_WIDGET/DBCONNECTION
EntityInfo entityInfo=new EntityInfo(reportId,"REPORT");
```

6. Create object of EntityAccessRight and set accessrights

```
EntityAccessRight ear=new EntityAccessRight();
```

```
ear.setAppInfo(appInfo);
/*     access level can have below possible values-
FULLACCESS, PARTIALACCESS, DENYACCESS, NONEACCESS
  * public void setAccessLevel(int accessLevel)
  * @param accessLevel : access level can be 0/1/2/3
*/
ear.setAccessLevel(Enums.SecurityTypes.AccessLevel.PARTIALACCESS)
;
ear.setAccessRightGrants("0,2,4,6,7,8,12");

ear.setOpCode("REPLACE");
```

Method: grantEntityPrivileges

This API allows the user to assign access rights information to a user/role/organization or Everyone on an entity.

Syntax

```
public void grantEntityPrivileges(EntityInfo entity,
```

```
EntityAccessRight entityAccessRight,
UserInfo userInfo)throws ISecurityException.
```

Parameters:

- **entity -** The entity object.This object must be created by setting entityId and entityType. entityTypes supported are defined in EntityTypeNames class..
- **entityAccessRight:** The entityAccessRight object. This object should contain the AppInfo object with the credentials of the user to which grants are to be assigned and access level as defined in Enums.SecurityTypes.AccessLevel
- **UserInfo:** Details of current user who is providing access rights.
- **Throws**: ISecurityException: - If the request cannot be performed successfully. This happens - 1.if connection can't be established with the engine or 2.if read or write operation cant be performed from or to the engine. 3.If some error has occured while executing the request on the engine. 4.If the response xml obtained from server cannot be parsed

Example

Given below is the example of actual implementation of this method:

```
sMgr.grantEntityPrivileges(entityInfo, ear, requestorUserInfo);
```

Assign Entity Privileges to User

This Java API is used to assign the Entity privileges to the specific User.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/User Management/ AssignEntityPrivilegesToUser.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a SecurityManager class object for getting the controller information for all Administration related operations.

```
SecurityManager sMgr=SecurityManager.getInstance();
```

4. Create instance of AppInfo that contains the UserId, Organization of the user to which the access rights are to be assigned.
   Also set AppType="USER"

```
String userId="Mary";
String OrgId="TestOrg";
```

```
  //AppType for which the privileges are to be set, "" for Org,
"USER" for User and "ROLE" for Role
  String appType = "USER";

//Create instance of AppInfo that contains entity for which
access rights are to be assigned
AppInfo appInfo = new AppInfo(userId,OrgId,appType);
```

5. Create instance of EntityInfo which contains entity whose access rights are assigned on Category.

```
// Query ID whose System Privileges are to be set for the
specified user.
String entityID="SalesQuery";

//Set desired entity : CATEGORY/REPORT/QO/PO/CUBEOBJECT
/DASHBOARD2/DASHBOARD_WIDGET/DBCONNECTION
EntityInfo entityInfo=new EntityInfo(entityID,"QO");
```

6. Create object of EntityAccessRight and set accessrights

```
EntityAccessRight ear=new EntityAccessRight();
```

```
ear.setAppInfo(appInfo);
/*     access level can have below possible values-
FULLACCESS, PARTIALACCESS, DENYACCESS, NONEACCESS
  * public void setAccessLevel(int accessLevel)
  * @param accessLevel : access level can be 0/1/2/3
*/
ear.setAccessLevel(Enums.SecurityTypes.AccessLevel.PARTIALACCESS)
;
ear.setAccessRightGrants("0,2");
```

Method: grantEntityPrivileges

This API allows the user to assign access rights information to a user/role/organization or Everyone on an entity.

Syntax

```
public void grantEntityPrivileges(EntityInfo entity,
EntityAccessRight entityAccessRight,
UserInfo userInfo)throws ISecurityException.
```

Parameters:

- **entity -** The entity object.This object must be created by setting entityId and entityType. entityTypes supported are defined in EntityTypeNames class..
- **entityAccessRight:** The entityAccessRight object. This object should contain the AppInfo object with the credentials of the user to which grants are to be assigned and access level as defined in Enums.SecurityTypes.AccessLevel
- **UserInfo:** Details of current user who is providing access rights.
- **Throws**: ISecurityException: - If the request cannot be performed successfully. This happens - 1.if connection can't be established with the engine or 2.if read or write operation cant be performed from or to the engine. 3.If some error has occured while executing the request on the engine. 4.If the response xml obtained from server cannot be parsed

Example

Given below is the example of actual implementation of this method:

```
sMgr.grantEntityPrivileges(entityInfo, ear, requestorUserInfo);
```

Assign System Privileges to the User

This Java API is used to assign the System Privileges to the User. The user would require to provide its identity i.e. user Info as well as of the user who is providing System Privileges.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/User Management/ AssignSystemPrivilegesToUser.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a SecurityManager class object for getting the controller information for all Administration related operations.

```
SecurityManager sMgr=SecurityManager.getInstance();
```

4. Create instance of User for which System Privileges are to be set.

```
String userId="FinanceUser";
String organization=="MyOrg";
UserInfo userInfo = sMgr.getUserById(userId, organization,
requestorUserInfo);
```

5. Assign System Privileges to assigneeUser.

Method : setSystemPrivileges

```
public                                                          void
setSystemPrivileges(java.lang.String systemPrivileges)
```

Enums.SecurityTypes.SystemPrivileges.*CATEGORY_SETUP*
Enums.SecurityTypes.SystemPrivileges.*DATA_ADMIN*
Enums.SecurityTypes.SystemPrivileges.*IM_SUPPORT*
Enums.SecurityTypes.SystemPrivileges.systemPrevilegesMap
Enums.SecurityTypes.SystemPrivileges.*SCHEDULER*
Enums.SecurityTypes.SystemPrivileges.*REPORT_DESIGNER*
Enums.SecurityTypes.SystemPrivileges.*DEPLOYREPORTBUNDLE*
Enums.SecurityTypes.SystemPrivileges.*CATEGORY_SETUP_GLOBAL*
Enums.SecurityTypes.SystemPrivileges.*SCHEDULER_GLOBAL*
Enums.SecurityTypes.SystemPrivileges.*DATA_ADMIN_GLOBAL*
Enums.SecurityTypes.SystemPrivileges.*ADHOCREPORTDESIGNER*
Enums.SecurityTypes.SystemPrivileges.*WIDGET_DESIGNER*
Enums.SecurityTypes.SystemPrivileges.*GENERATE_LINK*
Enums.SecurityTypes.SystemPrivileges. *GENERATE_LINK_GLOBAL*

This method sets System Privileges in the comma-separated format.

```
userInfo.setSystemPrivileges(new
com.intellica.client.utils.Utility().getRightsFromMaskedValue(
com.intellica.client.common.Enums.SecurityTypes.SystemPrivileges.
CATEGORY_SETUP |
com.intellica.client.common.Enums.SecurityTypes.SystemPrivileges.
SCHEDULER |
com.intellica.client.common.Enums.SecurityTypes.SystemPrivileges.
SCHEDULER_GLOBAL |
com.intellica.client.common.Enums.SecurityTypes.SystemPrivileges.
WIDGET_DESIGNER |
com.intellica.client.common.Enums.SecurityTypes.SystemPrivileges.
REPORT_DESIGNER |
com.intellica.client.common.Enums.SecurityTypes.SystemPrivileges.
ADHOCREPORTDESIGNER |
Enums.SecurityTypes.SystemPrivileges.GENERATE_LINK));

sMgr.modifyUser(userInfo,requestorUserInfo);
```

Assign Role To User

This Java API is used to assign a specified existing role to the application user.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/User Management/ AssignRoleToUser.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a SecurityManager class object for getting the controller information for all Administration related operations.

```
SecurityManager sMgr=SecurityManager.getInstance();
```

4. Create instance of User for which System Privileges are to be set.

```
String userId ="User1"; // Id of the application user
String orgId ="MyOrg";

UserInfo userInfo=sMgr.getUserById(userId, orgId
,requestorUserInfo);
```

5. Create instance of RoleInfo that is to be assigned to the User.

```
String roleId ="Manager"; // Id of the role to be Assigned
RoleInfo roleInfo  =  sMgr.getRoleById(roleId, orgId
,requestorUserInfo);
```

6. Grant this Role Info to the assignee User.

Method: grantRoleToUser

```
public void grantRoleToUser( targetUInfo,
                             rInfo,
 userInfo)
                     throws
```

This method grants new role to User and keeps previous roles of that user intact.

Parameters:

- **TargetUInfo:** com.intellica.client.common.UserInfo class object to which roles has to be assigned.
- **RInfo:** com.intellica.client.security.RoleInfo object to be granted.
- **UserInfo:** The details of the requesting user.

```
sMgr.grantRoleToUser(userInfo,roleInfo,requestorUserInfo);
```

Assign Roles To User

This Java API is used to assign multiple roles to the application user.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/User Management/ AssignRolesToUser.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a SecurityManager class object for getting the controller information for all Administration related operations.

```
SecurityManager sMgr=SecurityManager.getInstance();
```

4. Create instance of User for which System Privileges are to be set.

```
String userId="TestUser";// Id of the application user
String orgId="MyOrg";
UserInfo userInfo=sMgr.getUserById(userId, orgId
,requestorUserInfo);
```

5. Create instance of RoleInfo that is to be assigned to the User.

```
String roleId1 = "Manager";//Id of the role1 to be Assigned
String roleId2 = "Administrator";//Id of the role2 to be assigned
RoleInfo roleInfo1  =  sMgr.getRoleById(roleId1, orgId
,requestorUserInfo);
RoleInfo roleInfo2  =  sMgr.getRoleById(roleId2, orgId
,requestorUserInfo);
```

6. Grant these Roles to the assignee User.

Method: assignRolesToUser

```
public void assignRolesToUser( targetUInfo,
                               java.util.ArrayList roleInfos,
                                userInfo)
                        throws
```

This method attaches a User with more than one role.

Parameters:

- **targetUInfo**: The user to be attached with a role.
- **roleInfos**: Array List of Roles to be attached with user.
- **userInfo**: The details of the current user.

```
ArrayList roleInfoArr = new ArrayList();
roleInfoArr.add(roleInfo1);//Add roleInfo object role1 to Array
roleInfoArr.add(roleInfo2);//Add roleInfo object "role2" to Array

sMgr.assignRolesToUser(userInfo,roleInfoArr,requestorUserInfo);
```

Revoke Role from User

This Java API is to revoke specified role from user

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/User Management/ RevokeRoleFromUser.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a SecurityManager class object for getting the controller information for all Administration related operations.

```
SecurityManager sMgr=SecurityManager.getInstance();
```

4. Get User by User Id whose role is to be revoked.

```
String userId = "User1"; // Id of the application User
String orgId= "MyOrg";
UserInfo userInfo=sMgr.getUserById(userId, orgId
,requestorUserInfo);
```

5. Create instance of RoleInfo that is to be revoked for the given User.

```
String roleId = "Manager"; // Id of the role to be Assigned
RoleInfo roleInfo=sMgr.getRoleById(roleId, orgId
,requestorUserInfo);
```

6. Revoke the Role from assignee User.

Method: revokeRoleFromUser

```
public void revokeRoleFromUser(UserInfo targetUInfo,
                               RoleInfo rInfo,
                               UserInfo userInfo)
                        throws ISecurityException
```

This method revokes or removes role assigned previously to the user.

Parameters:

- **TargetUInfo:** com.intellica.client.common.UserInfo class object to which role has to be revoked.
- **rInfo**: com.intellica.client.security.RoleInfo object to be revoked from the user.
- **userInfo**: The details of the current User.

```
sMgr.revokeRoleFromUser(userInfo,roleInfo,requestorUserInfo);
```

User Mapping

This Java API is used to map the host application user with Intellicus user in the Intellicus Repository.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/User Management/UserMapping.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a SecurityManager class object for getting the controller information for all Administration related operations.

```
SecurityManager sMgr=SecurityManager.getInstance();
```

4. Get User by User Id to which , new user is to be mapped

```
String userId="Smith";// Id of the application user
String password="";
String orgId="MyOrg";
UserInfo userInfo  = new UserInfo(userId, password , orgId);
```

5. Map the new User implementing mapAppIdToUser method.

```
String newUserId="John";
sMgr.mapAppIdToUser(newUserId,userInfo, requestorUserInfo);
```

Details of Method: mapAppIdToUser

```
public void mapAppIdToUser(java.lang.String appId,
                           UserInfo targetUInfo,
                           UserInfo userInfo)
                    throws ISecurityException
```

This method adds a mapping between an AppId and the user.

Parameters:

- **AppId:** is the application Id.
- **TargetUInfo:** is the targeted User.
- **UserInfo:** the current user details.

Delete User

This Java API is used to delete an existing user with given userId from the Intellicus Repository.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/User Management/ DeleteUser.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a SecurityManager class object for getting the controller information for all Administration related operations.

```
SecurityManager sMgr=SecurityManager.getInstance();
```

4. Get UserInfo instance of the user that is to be deleted.

```
String userId="FinanaceUser";
String orgId="FinanceOrg";//Organization Id in which user
                          exists.
UserInfo targetUserInfo=sMgr.getUserById(userId,orgId,
                                   requestorUserInfo);
```

5. Delete the User.

Method: deleteUser

```
public void deleteUser(UserInfo targetUInfo,
                       UserInfo userInfo)
                throws ISecurityException
```

This method deletes an existing User's Info at the Report Engine.

Parameters:

- **TargetUInfo:** The UserInfo object of the user to be deleted.
- **UserInfo:** The details of the current user.

```
sMgr.deleteUser(targetUserInfo,requestorUserInfo);
```

Role

Get Role List

This Java API is used to get the list of all Roles present in Intellicus Repository.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/Role Management/ GetRoleList.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a SecurityManager class object for getting the controller information for all Administration related operations.

```
SecurityManager sMgr=SecurityManager.getInstance();
```

4. Get the role list.

Method: getRoleList

```
public java.util.ArrayList getRoleList(UserInfo userInfo)
                                throws ISecurityException
```

This method returns the list of all roles created in Intellicus repository.

Returns:

ArrayList of roleInfo Bean class RoleInfo

```
roleList=sMgr.getRoleList(requestorUserInfo);
```

Create Role (Add Role)

This Java API is used to add a new role in the existing Organization.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/Role Management/ AddRole.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.

3. Create a SecurityManager class object for getting the controller information for all Administration related operations.

```
SecurityManager sMgr=SecurityManager.getInstance();
```

4. Create instance of RoleInfo for the role to be added in the Organization.

```
String roleName = "MANAGER";
String orgId = "Org1";
RoleInfo newRole = new RoleInfo(roleName, orgId );
```

5. Add the role in the organization.

Method : addRole

```
public void addRole(RoleInfo roleInfo,
                    UserInfo userInfo)
             throws ISecurityException
```

This method adds a new Role at the Report Engine.

Parameters:

- **roleInfo:** The details of new Role.
- **userInfo:** The current user.

```
sMgr.addRole(newRole,requestoruserInfo);
```

Assign Category Privileges To Role

This Java API is used to assign the Access privileges of a Category to the existing Role.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/Role Management/ AssignCategoryPrivilegesToRole.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a SecurityManager class object for getting the controller information for all Administration related operations.

```
SecurityManager sMgr=SecurityManager.getInstance();
```

4. Create EntityInfo and AppInfo

```
//RoleID of the Role for which AccessRights are to be assigned
```

```
String roleId     ="Manager";//role  Id  to  be  checked  whether
exists or not in the Intellicus server
String orgId    ="hostOrg";
String categoryId ="B2996F87-FE41-8D19-6C67-B3C22803DD99";

EntityInfo entityInfo=new EntityInfo(categoryId,"CATEGORY");
AppInfo appInfo = new AppInfo(roleId,orgId,"ROLE");
```

5. Create EntityAccessRight Object and set this AppInfo and EntityInfo in its object.

```
EntityAccessRight ear=new EntityAccessRight();

/*public void setAppInfo(AppInfo appInfo)Parameters:
 *appInfo - : AppInfo object.
 * */
ear.setAppInfo(appInfo);
/*     access level can have three values possible.
 *     Deny Access      = 0
 *  Full Access         = 1
 *     Partial Access   = 2
 *  None Access    = 3*/
/**
 * public void setAccessLevel(int accessLevel)
 * @param accessLevel : access level can be 0/1/2
 */
ear.setAccessLevel(Enums.SecurityTypes.AccessLevel.FULLACCESS);
```

6. Assign Category Privileges to assigneeUser using assignCategoryPrivilegesToUser.

Method: grantEntityPrivileges

Syntax

```
public void grantEntityPrivileges(EntityInfo entity,
EntityAccessRight entityAccessRight, UserInfo userInfo)
```

API allows the user to assign access rights information to a user/role/organization or Everyone on an entity.

Parameters:

- **entity:** The entity object.This object must be created by setting entityId and entityType. entityTypes supported are defined in EntityTypeNames class.

- **entityAccessRight:** The entityAccessRight object. This object should contain the AppInfo object with the credentials of the user to which grants are to be assigned and access level as defined in Enums.SecurityTypes.AccessLevel
- **userInfo:** Details of current user who is providing access rights.

Example

Given below is the example of actual implementation of this method:

```
sMgr.grantEntityPrivileges(entityInfo, ear, adminUserInfo);
```

Assign Report Privileges to Role

This Java API is used to assign the Access privileges for a Report to the Role.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/Role Management/ AssignReportPrivilegesToRole.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a SecurityManager class object for getting the controller information for all Administration related operations.

```
SecurityManager sMgr=SecurityManager.getInstance();
```

4. Create EntityInfo and AppInfo

```
//RoleID of the Role for which AccessRights are to be assigned
String roleId="Manager";
String organisationId="Intellica";
String reportId="638CB7A6-F504-0ECD-008F-10B25253DCA8";

EntityInfo entityInfo=new EntityInfo(reportId,"REPORT");
AppInfo appInfo = new AppInfo(roleId,organisationId,"ROLE");
```

5. Create EntityAccessRight Object and set this AppInfo and EntityInfo in its object.

```
EntityAccessRight ear=new EntityAccessRight();

/*public void setAppInfo(AppInfo appInfo)Parameters:
 *appInfo - : AppInfo object.
 * */
ear.setAppInfo(appInfo);
/*     access level can have three values possible.
```

```
 *      Deny Access      = 0
 *   Full Access         = 1
 *      Partial Access   = 2
 *   None Access    = 3*/
/**
 * public void setAccessLevel(int accessLevel)
 * @param accessLevel : access level can be 0/1/2
 */
ear.setAccessLevel(Enums.SecurityTypes.AccessLevel.FULLACCESS);
```

6. Assign Category Privileges to assigneeUser using assignCategoryPrivilegesToUser.

Method: grantEntityPrivileges

Syntax

```
public void grantEntityPrivileges(EntityInfo entity,
EntityAccessRight entityAccessRight, UserInfo userInfo)
```

API allows the user to assign access rights information to a user/role/organization or Everyone on an entity.

Parameters:

- **entity:** The entity object.This object must be created by setting entityId and entityType. entityTypes supported are defined in EntityTypeNames class.
- **entityAccessRight:** The entityAccessRight object. This object should contain the AppInfo object with the credentials of the user to which grants are to be assigned and access level as defined in Enums.SecurityTypes.AccessLevel
- **userInfo:** Details of current user who is providing access rights.

Example

Given below is the example of actual implementation of this method:

```
sMgr.grantEntityPrivileges(entityInfo, ear, adminUserInfo);
```

Delete Role

This Java API is used to delete a particular Role of an existing Organization in Intellicus Repository.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/Role Management/ DeleteRole.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a SecurityManager class object for getting the controller information for all Administration related operations.

```
SecurityManager sMgr=SecurityManager.getInstance();
```

4. Create instance of RoleInfo for the role which is to be deleted from the Organization.

```
//Role ID i.e. to be deleted
String roleId     ="Manager";// Id of the role to be deleted
//Organization from which the given role is to be deleted.
String orgId ="Test";

RoleInfo roleInfo=new RoleInfo(roleId,orgId);
```

5. Assign the Report Privileges to the Role for given Report Id in a particular Category.

Method: deleteRole

```
public void deleteRole(RoleInfo roleInfo,
                       UserInfo userInfo)
                throws ISecurityException
```

This method deletes a Role at the Report Engine.

Parameters:

- **RoleInfo:** The role to be deleted.
- **userInfo**: The current user details.

```
sMgr.deleteRole(roleInfo,requestorUserInfo);
```

Report Management Actions

Categories

Import

```
import com.intellica.client.common.UserInfo;

import com.intellica.client.layout.LayoutManager;
import com.intellica.client.reportutils.Category;
import com.intellica.client.exception.
         LayoutHandlerException;

import java.util.Vector;
import java.util.HashMap;
```

Get Category List

This Java API retrieves the list of all categories from the Intellicus repository.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/Category Management/ GetCategoryList.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a Layout Manager class object for report layout management related operations.

```
LayoutManager lm= new LayoutManager();
```

4. Get the list of all Categories in Repository.

Method: **getCategoryList**

```
public                                              java.util.Vector
getCategoryList(UserInfo requestorUserInfo)
          throws LayoutHandlerException
```

This method returns a vector of Category objects. The method requests report server and returns the report categories present in the report repository, both public and private to requestor user. The public categories will be restricted to those categories that the requestorUserInfo has access to read. Each category object provides getIsPublic boolean method to identify the scope of category.

Parameters:

RequestorUserInfo: UserInfo for authorization. Pass getEmptyInstance() in case of report server runs in Security-Off mode.

Returns:

Vector of Category objects.

```
Vector vecCatList= new Vector();
vecCatList=lm.getCategoryList(requestorUserInfo);
```

The other three APIs for getting Category List are-

1. Returns a vector of categories from the report server on the basis of filters

```
public Vector getCategoryList(Filter filter,UserInfo userInfo)
```

2. Returns a vector of report categories from the report server with given access rights

```
public Vector getCategoryList(int accessRight, UserInfo userInfo)
```

3. Returns a vector of categories from the report server on the basis of filters

```
public Vector
getCategoryList(com.impetus.intera.layout.adhoc.data.Filters
filters,UserInfo userInfo)
```

All these returns Vector containing Category objects.

Add a new Category

This Java API is used to add a new Category in the Intellicus Repository.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/Category Management/ CreateNewCategory.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a Layout Manager class object for report layout management related operations.

```
LayoutManager lm= new LayoutManager();
```

4. Create an instance of Category i.e to be added in Repository and set its various attributes of Categories.

```
String catName="NewDemoCategory";
String catDescription = "This is the new category just created";
String catId = "123456";
Category newCategory=new Category(catName);
//Set the various properties of the Category
newCategory.setIsPublic(true);
newCategory.setIsHidden(false);
newCategory.setDescription(catDescription);
//To set the category id
newCategory.setCategoryId(catId);
newCategory.setMenuName(catName);
```

5. Add this Category in the Repository.

Method: **addCategory**

```
public Category addCategory(Category category,
                            UserInfo userInfo)
                     throws LayoutHandlerException
```

This method adds new category to the Intellicus repository.

```
//Adds new category to the Intellicus repository.
lm.addCategory(newCategory,requestorUserInfo);
```

Delete Category

This Java API is used to delete an Existing Category from the Intellicus repository along with the Reports present in that Category.User need to provide the Category ID of the Category which is to be deleted.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/Category Management/ DeleteCategory.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a Layout Manager class object for report layout management related operations.

```
LayoutManager lm= new LayoutManager();
```

4. Delete the Category with given Category Id.

This method is used to delete the existing category from the report engine's report layout repository along with the reports contained in it.

Method: deleteCategory

```
public void deleteCategory(java.lang.String catId,
                           boolean unConditional,
                           UserInfo userInfo)
                    throws LayoutHandlerException
```

Parameters:

- **CatId:** Category ID of the Existing Category that is to be deleted.
- **UnConditional:** If true than deletes the category even if report layouts are present in the category along with all its reports.
- **UserInfo:** UserInfo object authorization.

```
//Category ID of the Category that is to be deleted.
  String categoryID="DemoCategory";

lm.deleteCategory(categoryID, true, requestorUserInfo);
```

GetSubCategories

This Java API is used to delete an Existing Category from the Intellicus repository along with the Reports present in that Category.User need to provide the Category ID of the Category which is to be deleted.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/Category Management/DeleteCategory.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a Layout Manager class object for report layout management related operations.

```
LayoutManager lm= new LayoutManager();
```

4. Get the sub categories for given Category Id.

This method is used to delete the existing category from the report engine's report layout repository along with the reports contained in it.

Method: getSubCategories

```
public LinkedHashMap getSubCategories()
```

```
throws LayoutHandlerException
```

Parameters:

- **CatId:** Category ID of the Category whose categories are to be obtained
- **UserInfo:** UserInfo object authorization.

```
//Category ID of the Category that is to be deleted.
String categoryID="DemoCategory";
categoryObj = lm.getCategory(categoryID, requestorUserInfo);

//now, fill the sub-categories in the category object.
Filter filter = new Filter();
filter.setFilterField("ENTITYTYPE", "CATEGORY");

lm.populateEntityListForCategory(categoryObj,filter,requestorUser
Info);
LinkedHashMap subcategories = categoryObj.getSubCategories();
```

The other related APIs for Category Management are-

1. Public void **deleteSubCategory**(String subCategoryId)
2. public void **setSubCategories**(LinkedHashMap subCategories)
3. public Category **getParentCategory**()
4. public void **setParentCategory**(Category parentCategory)

Report Operations

Import

```
import com.intellica.client.common.UserInfo;

import com.intellica.client.layout.LayoutManager;
import com.intellica.client.reportutils.Report;
import com.intellica.client.exception.
         LayoutHandlerException;

import java.util.Vector;
import java.util.HashMap;
```

Run Report

This Java API is used to execute a Report with given Report Id.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/Report Operations/ RunReport.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create an instance of reportExecuter i.e. a class having methods for executing a report.

```
ReportExecutor report = new ReportExecutor();
```

4. Set the System Parameters required for running the Report.

```
//Instantiate a collection object for setting system parameters
HashContainer hcSysParams = new HashContainer();

//fill up the system parameter values
//Report ID which is required to run
String reportId    = "Customer Report";

// This will set Id of the report.
hcSysParams.put("REPORT_ID",reportId);
// In database mode only reportId needed.don't prefix //category
id
// This will set the output format for the report.report //format
can be PDF/XLS/HTML

  hcSysParams.put("REPORT_FORMAT",InteraConstants.ReportFormats.P
DF);
```

5. Set the User/Business Parameters

```
//Instantiate a collection object for setting user/business
//parameters
HashContainer hcUserParams =new HashContainer();
String countryName = "Australia";
String roleId = "3818";
// fill up the user params if report contains any parameter
hcUserParams.put("Country", countryName);
// Parameter Name as created in the report layout(IRL)
hcUserParams.put("Role_ID", roleId);
```

6. Instantiate the Byte Array Output Stream object.

```
ByteArrayOutputStream reportData = new ByteArrayOutputStream();
```

7. Run the Report.

Method: **runReport**

```
public void runReport(UserInfo userInfo,
                      HashContainer systemParameterHash,
                      HashContainer userParameterHash,
                      IStreamCallback pCallback,
                      java.io.ByteArrayOutputStream reportData,
                      boolean isPublic)
               throws ClientException
```

This method executes a report by taking the values of system-defined parameters (and report parameters) in parameters as HashContainer and gives the generated report data in the passed outputStream. This method generates the complete report, and then returns the control to the calling code. In this case, the calling code must catch the exceptions right at the calling place. The calling code can use the output stream after the method call.

Parameters:

- **UserInfo:** userInfo object will be used for passing user information to the Report Engine. Can be left as null when application is not using Intellica Security Module.
- **SystemParameterHash:** HashContainer instance, Contains the values of system parameters.
- **UserParameterHash:** HashContainer instance, Contains the values of Report parameters. Can be left as null when there are no parameters in the report.
- **PCallback:** IStreamCallback instance, For future usage.
- **ReportData: T**he RefByteArrayOutputStream instance required for the report output.The report output in the form of byte array is added in this output stream.

```
report.runReport(requestorUserInfo,hcSysParams,hcUserParams,null,
reportData,true);
```

Get the list of Published Reports of a particular report

This Java API is used to get the list of all published reports corresponding to given Report Id.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/Report Operations/ GetPublishedReport.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a Layout Manager class object for report layout management related operations.

```
LayoutManager lm= new LayoutManager();
```

4. Set the filter for getting Saved Report List.

The various filter settings could be:

```
Enums.Filters.SavedReportList.CATID
Enums.Filters.SavedReportList.ENTITY_PROPERTIES
Enums.Filters.SavedReportList.FROMDATE
Enums.Filters.SavedReportList.GENERATOR_ID
Enums.Filters.SavedReportList.GENERATOR_TYPE
Enums.Filters.SavedReportList.ISPUBLIC
Enums.Filters.SavedReportList.ORGID
Enums.Filters.SavedReportList.ORPHANED
Enums.Filters.SavedReportList.PUBLISHSTATUS
Enums.Filters.SavedReportList.REPORTID
Enums.Filters.SavedReportList.WORKFLOW_STATE
Enums.Filters.SavedReportList.TODATE
Enums.Filters.SavedReportList.USERID
```

```
Filter filterObj = new Filter();
String reportId="SalesReport";
filterObj.setFilterField(Enums.Filters.SavedReportList.REPORTID,r
eportId);
```

5. Get the List of Saved Reports based on the filters applied.

Method: getViewSavedReport

```
public void getViewSavedReport(SavedReport savedReport,
java.io.ByteArrayOutputStream reportData,
                               HashContainer sysParams,
```

```
 userInfo)
                              throws LayoutHandlerException
```

This method is used to get the saved report with in the ByteArrayOutputStream passed as a parameter.

Parameters:

- **SavedReport:** Saved report object SavedReport.
- **ReportData:** ByteArrayOutputStream contains the Report data.
- **SysParams:** HashContainer for report system parameters.
- **UserInfo:** UserInfo For authorization.

```
Vector vecPublishedReportList =
lm.getSavedReportList(filterObj,requestorUserInfo);
```

Report Layout Management

Import

```
import com.intellica.client.common.UserInfo;

import com.intellica.client.layout.LayoutManager;
import com.intellica.client.reportutils.Report;
import com.intellica.client.exception.
         LayoutHandlerException;

import java.util.Vector;
import java.util.HashMap;
```

GetAllReportList

This Java API is used to get the complete list of all the Reports from the Intellicus Repository.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/Report Layout Management/ GetAllReportList.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a Layout Manager class object for report layout management related operations.

```
LayoutManager lm= new LayoutManager();
```

4. Get the Report List

Method : getReportList

```
public java.util.Vector getReportList(UserInfo userInfo)
                                  throws LayoutHandlerException
```

Returns a vector of all report names from the report server The list will be restricted to accessible reports by this user (Applies to report server enterprise edition only)

Parameters:

- **UserInfo:** UserInfo for authorization.

Returns:
Vector of all report names from the report server.

Part of Sample Code implementing "getReportList":

```
//Vector that will be used to store the Report List
Vector vecReportList= new Vector();

//This method returns a vector of all report names from the
report server
vecReportList=lm.getReportList(requestorUserInfo);
```

The other APIs for getting Report List are-

1. Returns a vector of report objects from the report server on the basis of requesting filters.

```
public Vector getReportList(Filter,UserInfo)
```

2. Returns a vector of all report names from the report server. The list will be restricted to accessible reports by this user

```
public Vector getReportList(int categoryAccessRight, int
reportAccessRight,UserInfo userInfo)
```

3. Returns a vector of all report names from the report server. The list will be restricted to accessible reports by this user

```
public Vector getReportList(int reportAccessRight,UserInfo
userInfo)
```

4. This method gets the list of reports based on filters (request details) specified in ReportListRequest

```
public Vector getReportList(ReportListRequest
reportListRequest,UserInfo userInfo)
```

MoveReport

This Java API is used to move an existing report to another Destination Category.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/Report Layout Management/ MoveReport.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.

3. Create a Layout Manager class object for report layout management related operations.

```
LayoutManager lm= new LayoutManager();
```

4. Pass the Source ReportId which is to be copied and the destination category Id , to which it is copied so as to move the Report.

Method : moveReport

```
public void moveReport(java.lang.String sourceReportId,
                       java.lang.String destinationCategoryId,
                       Report report,
                       UserInfo userInfo)
                throws LayoutHandlerException
```

Method to Move a report to the new category.

Parameters:

- **SourceReportId:** source report id, mandatory argument for moving the report.
- **DestinationCategoryId:** destination category id,where new report has to be moved.if this is null then report objects's category id will be taken as destination category.
- **Report**: instance of Report class for move.
- **UserInfo:** UserInfo object for authorization.

```
//ReportID of the Report that is to be moved to new Category
String reportID="234CBC33-EC19-E754-7490-43DA2A24728C";

//CategoryID of the Destination Category where the report needs
       to be moved
String destCategoryID="61FCA109-7E0D-D023-A19C-A054A38FC9B6";

//Method to copy the report to anther Category

lm.moveReport(reportID, destCategoryID, r,requestorUserInfo);
```

Get Report List For Category

This Java API is used to get the complete list of Reports in given Category.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/Report Layout Management/ GetReportListForCategory.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a Layout Manager class object for report layout management related operations.

```
LayoutManager lm= new LayoutManager();
```

4. Pass the Category Id whose Reports are to be listed.

Method : getReportListForCategory

```
public java.util.Vector getReportListForCategory
       (java.lang.String categoryId,
        UserInfo requestorUserInfo)
throws LayoutHandlerException
```

Method returns a vector of Report objects. The method requests report server and returns the reports present in the report repository under given report category, both public and private to requestor user. The public reports will be restricted to those reports that the requestorUserInfo has access to read. Each report object provides getIsPublic boolean method to identify the scope of report.

Parameters:

- **CategoryId:** The category Id of the category from which, the report layout list is requested. Intellicus allows category ids upto 256 charecters long.
- **RequestorUserInfo:** UserInfo for authorization. Pass getEmptyInstance() in case of report server runs in Security-Off mode.

Returns:
Vector of Reports.

```
Vector vecCatList= new Vector();
String categoryId = "01-Sales";
vecCatList=lm.getReportListForCategory
                 (categoryId , requestorUserInfo);
```

The other related APIs for getting Report List For Category are-

1. Returns a vector of all report names from the report server.The list will be restricted to accessible reports by this user.

```
public       Vector       getReportListForCategoryAccessRight(int
categoryAccessRight,UserInfo userInfo)
```

2. Returns a vector of all report names from the report server.The list will be restricted to accessible reports by this user.

@**param** categoryId: The category Id for which the report layout list is requested.
@**param** reportAccessRight masked value of the access right.

```
public Vector getReportListForCategoryWithAccessRights(String categoryId,int categoryAccessRight, int reportAccessRight, UserInfo userInfo)
```

3. Returns a vector of all report names from the report server.The list will be restricted to accessible reports by this user.
   @**param** categoryId: The category Id for which the report layout list is requested.
   @param categoryAccessRight masked value of the access right.

```
public Vector getReportListForCategoryWithCategoryRight(String categoryId,int categoryAccessRight,UserInfo userInfo)
```

4. The method requests report server and returns the reports present in the report repository under given report category, both public and private to requestor user.

```
public Vector getReportListForCategoryWithReportRight(String categoryId, int reportAccessRight,UserInfo userInfo)
```

Get Saved Report List

This Java API is used to get the list of Saved Reports in a particular Category/Report.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/Report Layout Management/GetSavedReportList.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a Layout Manager class object for report layout management related operations.

```
LayoutManager lm= new LayoutManager();
```

4. Pass the Category Id/Report Id whose Saved Reports are to be listed.

Method : getSavedReportList

```
public java.util.Vector getSavedReportList
       (Filter filter, UserInfo requestorUserInfo)
throws LayoutHandlerException
```

Method returns a vector of Report objects. The method requests report server and returns the saved reports present in the report repository under given report category/Report, both public and private to requestor user.

Parameters:

- **filter:** Filter can set the values of CATID and REPORTID.
- **RequestorUserInfo:** UserInfo for authorization.

Returns:

Vector of Saved Reports.

```
//Cat ID of the Category whose Reports are required to be Listed
String catId = "4F9245A7-D639-4F99-604D-F32641B77725"; //category
id to be set in filter
String reportId="All Country Sales - Linked Prompt";//Report Id
to be set in filter

Filter filter = new Filter();

  filter.setFilterField(com.intellica.client.common.Enums.Filters
.SavedReportList.CATID, catId);

  filter.setFilterField(com.intellica.client.common.Enums.Filters
.SavedReportList.REPORTID, reportId);
```

The other related APIs fro Saved Reports are:

1. public LinkedHashMap **getSavedReportsMap**()
2. public void **setSavedReportsMap**(LinkedHashMap savedReportsMap)
3. public void **deleteSavedReport**(String reportOId,String reportOutputId, UserInfo userInfo)
4. public void **updateSavedReport**(SavedReport savedReport, UserInfo userInfo)
5. public void **addSavedReport**(SavedReport savedReport)

Add Report Layout to Category

This Java API is used to add a report to an existing Category in the Intellicus repository. This would require User to provide the Category ID, and the details related to the Report i.e. to be added.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/Report Layout Management/ AddReportLayoutToCategory.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a Layout Manager class object for report layout management related operations.

```
LayoutManager lm= new LayoutManager();
```

4. Create an instance of Report i.e. to be added in the Category. Also, set the various attributes of report.

```
String file = "C:/UploadAll Country Sales.irl";

//Read the contents from the given file
InputStream is = new FileInputStream(file);

// Get the size of the file
long length = file.length();

if (length > Integer.MAX_VALUE) {
// File is too large
  throw new Exception("File is too Large");
}
else {
// Create the byte array to hold the data
byte[] bytes = new byte[(int) length];

// Read in the bytes
int offset = 0;
int numRead = 0;

while (offset < bytes.length  && (numRead = is.read(bytes, offset,
bytes.length- offset)) >= 0)
{
offset += numRead;
}
//Ensure all the bytes have been read in
if (offset < bytes.length) {
throw new IOException("Could not completely read the file ");
}
is.close();

String reportName = "Sample_Report";
String reportID = "Sample Report";

Report reportToAdd = new Report(reportName, reportID, bytes);
```

```
String categoryID = "Demo";
reportToAdd.setCategoryId(categoryID);
```

5. Upload the Report.

Method : uploadReport

```
public void uploadReport(Report report,
                         UserInfo userInfo)
                  throws LayoutHandlerException
```

This method uploads a report by taking the Report object as parameter. This method internally validates the Report object for report paramaters. The method also varifies if report already exists in the specified category.In case report object is not validated and varified the method throws LayoutHandlerException exception.The calling code must catch the exception right at the calling place.

Parameters:

- **Report:** Report class instance containing the report layout metadata and the byte array of report layout itself.
- **UserInfo:** UserInfo object use to validate the user against the application.

```
lm.uploadReport(reportToAdd, requestorUserInfo);
```

Copy Report

This Java API is used to copy an existing report to another destination category.

User needs to provide the ReportID that is to be copied and the destination category ID where it is to be copied.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/Report Layout Management/ CopyReport.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a Layout Manager class object for report layout management related operations.

```
LayoutManager lm= new LayoutManager();
```

4. Set the attributes for copied report.

```
Report r=new Report();
String rep_Name = "Copied_Report";
```

```
r.setMenuName(rep_Name);
```

5. Pass the Source Report Id to be copied and the destination Category where it is to be copied.

Method : **copyReport**

```
public void copyReport(java.lang.String sourceReportId,
                       java.lang.String destinationCategoryId,
                       Report report,
                       UserInfo userInfo)
                throws LayoutHandlerException
```

This method is used to make a copy of the existing report attributes.

Parameters:

- **SourceReportId:** source report id, mandatory argument for copying the report.
- **DestinationCategoryId:** destination category id, where new report has to be copied.if this is null then source report's category id will be taken as destination category.
- **Report:** instance of Report class for copy.
- **UserInfo:** UserInfo object for authorization.

Part of Sample Code implementing "copyReport":

```
String sourceReportId = "787F11B1-74E7-6792-9ACB-408753C0470F";
String destCategoryId = "61FCA109-7E0D-D023-A19C-A054A38FC9B6";
lm.copyReport(sourceReportId, destCategoryId, r,
requestorUserInfo);
```

Get Report Details

This Java API is used to get Report Details corresponding to the given Reoprt Id from the Intellicus Repository.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/Report Layout Management/ GetReportDetails.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a Layout Manager class object for report layout management related operations.

```
LayoutManager lm= new LayoutManager();
```

4. Get the Report Detail for given Report Id.

Method: **getReportDetails**

```
public Report getReportDetails(java.lang.String reportId,
                               UserInfo userInfo)
                        throws LayoutHandlerException
```

This method returns an object of Report class obtained from the report server.

Parameters:

- **Reported:** The report id whoes details should be obtained.
- **UserInfo:** UserInfo for authorization.

Returns:
An object of Report class obtained from the report server.

```
Report reportDetail=new Report();
String reportId = "91FEE269-3AEE-C23D-6F04-7A9979BEBE09";
reportDetail=lm.getReportDetails(reportId,requestorUserInfo);
```

Other related APIs for Report Layout Management are:

1. public **LinkedHashMap** getReportsMap()
2. public void **setReportsMap**(LinkedHashMap reportsMap)
3. public void **updateReport**(Report oldReport,Report newReport,UserInfo userInfo)
4. public void **updateReport**(Vector reportVector, UserInfo userInfo)
5. public void **addReport**(Report report)

Mass Operations

User is allowed to select more than one entity and perform the selected operation on those Entities. This could be done using below APIs.

Import

```
import com.intellica.client.common.UserInfo;

import com.intellica.client.layout.LayoutManager;
import com.intellica.client.reportutils.Report;
import com.intellica.client.exception.
         LayoutHandlerException;
```

```
import java.util.Vector;
import java.util.ArrayList;
import java.util.HashMap;
```

Copy Entities

This API is used to copy entities to another location. It returns modified object with new destination location.

Let's consider here, Queries are to be copied from one Category to Destination Category.

User needs to provide the Query IDs that are to be copied and the destination category ID where they are to be copied.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a Layout Manager class object for report layout management related operations.

```
LayoutManager lm= new LayoutManager();
```

4. Set the Query Ids that are to be copied.

```
String queryIds[] = new String
"1260772934500920312919291067188 2",
"1260351680968920312919291443149 3","1258703233232419216810228958 3
904"};
```

5. Create the EntityOperation object and set Query Entities for above given Query Ids and add in an ArrayList i.e. entityList.

```
ArrayList entityList = new ArrayList();
EntityOperation entityObj = null;

for(int i=0;i< queryIds.length;i++){
  entityObj = new EntityOperation();

  //Entity Type could be REPORT /CATEGORY /QUERY /DASHBOARD2/
  PARAMETER /SAVEDREPORT /DASHBOARD_WIDGET /CUBEOBJECT
  entityObj.setEntityType("QUERY");

  entityObj.setEntityId(queryIds[i]);
  entityList.add(entityObj);
}
```

6. Pass the arraylist of entities that are to be copied in the destination Category.

Method: **copyEntities**

```
public ArrayList copyEntities(ArrayList entityList, String
targetCatId, UserInfo userInfo)
throws LayoutHandlerException{
```

This method is used to copy entites in the destination Category.

Parameters:

- **entityList :** ArrayList of {@link com.intellica.client.common.EntityOperation} class instance.
- **targetCatId:** Destination location where this entity should be copied.
- **UserInfo:** UserInfo object for authorization.

Part of Sample Code implementing "copyEntities":

```
String destCategoryId = "DataLoader";

lm.copyEntities(entityList, destCategoryId, requestorUserInfo);
```

Delete Entities

This API is used to delete entities from the Repository.
Let's consider here, "Categories" are to be deleted from the Repository.

User needs to provide the Category IDs that are to be deleted.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a Layout Manager class object for report layout management related operations.

```
LayoutManager lm= new LayoutManager();
```

4. Set the Category Ids that are to be deleted.

```
String categoryIds[] = new String "Demo", "Sales_Category"};
```

5. Create the EntityOperation object and set Category Entities for above given Category Ids and add in an ArrayList i.e. entityList.

```
ArrayList entityList = new ArrayList();
EntityOperation entityObj = null;

for(int i=0;i< queryIds.length;i++){
  entityObj = new EntityOperation();

  //Entity Type could be REPORT /CATEGORY /QUERY /DASHBOARD2/
  PARAMETER /SAVEDREPORT /DASHBOARD_WIDGET /CUBEOBJECT
  entityObj.setEntityType("CATEGORY");

  entityObj.setEntityId(categoryIds [i]);
```

```
  entityList.add(entityObj);
}
```

6. Pass the arraylist of entities that are to be deleted from the Repository.

Method: **deleteEntities**

```
public ArrayList deleteEntities(ArrayList entityList, UserInfo
userInfo)
throws LayoutHandlerException{
```

This method is used to delete entities.

Parameters:

- **entityList:** ArrayList of {@link com.intellica.client.common.EntityOperation} class instance.
- **UserInfo:** UserInfo object for authorization.

Part of Sample Code implementing "copyEntities":

```
String destCategoryId = "DataLoader";

lm. deleteEntities(entityList, requestorUserInfo);
```

Following are the APIs related to Mass Operation-

1. public ArrayList **deleteEntities**(ArrayList entityList, UserInfo userInfo)
2. public ArrayList **copyEntities**(ArrayList entityList, String destinationCatId, UserInfo userInfo)
3. public ArrayList **copyLinkEntities**(ArrayList entityList,String destinationCatId, UserInfo userInfo)
4. public ArrayList **deLinkEntities**(ArrayList entityList, UserInfo userInfo)
5. public ArrayList **moveEntities**(ArrayList entityList, String destinationCatId, UserInfo userInfo)

Dashboards

Import

```
import com.intellica.client.common.UserInfo;

import com.intellica.client.dashboard.Dashboard;
import com.intellica.client.dashboard.DashboardManager;
import com.intellica.client.reportutils.Category;
import com.intellica.client.exception.
```

```
        LayoutHandlerException;

import java.util.Vector;
import java.util.HashMap;
```

Get Dashboard Details

This Java API is used to get the complete detail about the Dashboard like its Category, Description, Access Rights etc

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/DashBoard Management/ GetDashboardDetails.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a Dashboard Manager class object for Dashboard related settings.

```
DashboardManager dManager = new DashboardManager();
```

4. Get the Dashboard details whose Dashboard Id is given.

Method: **getDashBoardDetails**

```
public DashBoard getDashBoardDetails (String dashboardId,
UserInfo userInfo)
```

This method is used to get the requested DashBoard Object whose ID is passed as an argument.

Parameters:

param dashboardId: The unique identifier of the dashboard for which the details are to be obtained from the Report Server.

param userInfo: {@link com.intellica.client.common.UserInfo userInfo} object for authorization to get DashBoard.

Returns: Dashboard {@link com.intellica.client.dashboard.DashBoard2 DashBoard2} object containing dashboard details

```
Dashboard dashboard = dManager.getDashBoardDetails(DashboardId,
requestorUserInfo);
```

Get Dashboard List

This Java API is used to get the complete list of Dashboards from the Intellicus Repository for the Requestor user

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/DashBoard Management/ GetDashboardList.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a Dashboard Manager class object for Dashboard related settings.

```
DashboardManager dManager = new DashboardManager();
```

4. Get the Dashboard list.

Method: **getDashBoardList**

```
public DashBoard getDashBoardList (Filter filterObj,
UserInfo userInfo)
```

This method is used to get the list of Dashboards.

Parameters:

param filterObj: Filter object to set EntityType, CategoryId, Depth.

param userInfo: {@link com.intellica.client.common.UserInfo userInfo} object for authorization to get DashBoard.

Returns: Arraylist of Dashboards Objects

```
dashBoardList = dManager.getDashboardList(filterObj,
requestorUserInfo);
```

Get Dashboard Widget List

This Java API is used to get the complete list of the Dashboard widgets from the Intellicus Repository.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/DashBoard Management/ GetDashboardWidgetList.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a Dashboard Manager class object for Dashboard related settings.

```
DashboardManager dManager = new DashboardManager();
```

4. Get the Dashboard widget list whose Dashboard Id is given.

Method: **getDashboardWidgetList**

```
public ArrayList getDashboardWidgetList (Filter filterObj,
UserInfo userInfo)
```

Parameters:

param filterObj: Filter object to set EntityType DASHBOARD_WIDGET

param userInfo: {@link com.intellica.client.common.UserInfo userInfo} object for authorization to get DashBoard.

Returns: Arraylist of Dashboard Widgets

```
dashBoardWidgetList = dManager.getDashboardWidgetList(filterObj,
requestorUserInfo);
```

Get Dashboard Widgets for Category

This Java API is used to get the widgets present in given Category from the Intellicus Repository.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/DashBoard Management/ GetDashboardWidgetsForCategory.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a Dashboard Manager class object for Dashboard related settings.

```
DashboardManager dManager = new DashboardManager();
```

4. Get the Dashboard widget list whose Dashboard Id is given.

Method: **getDashboardWidgetList**

`public ArrayList getDashboardWidgetList (Filter` filterObj, `UserInfo` userInfo)

Parameters:

param filterObj: Filter object to set EntityType DASHBOARD_WIDGET

param userInfo: {@link com.intellica.client.common.UserInfo userInfo} object for authorization to get DashBoard.

Returns: Arraylist of Dashboard Widgets.

```
//To get the Dashboard list
filterObj.setFilterField(Enums.Filters.EntityList.ENTITYTYPE,Enum
s.Filters.EntityList.EntityType.DASHBOARD_WIDGET);

//To get Widget list for given Category.
filterObj.setFilterField(Enums.Filters.EntityList.CATEGORYID,
categoryId);

filterObj.setFilterField(Enums.Filters.EntityList.TRAVERSAL,Enums
.Filters.EntityList.Traversal.DOWN);

dashBoardWidgetList = dManager.getDashboardWidgetList(filterObj,
requestorUserInfo);
```

Delete Dashboard

This Java API is used to delete the Dashboard from the Intellicus Repository.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/DashBoard Management/ DeleteDashBoard.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a Dashboard Manager class object for Dashboard related settings.

```
DashboardManager dManager = new DashboardManager();
```

4. Delete the Dashboard whose Dashboard Id is given.

Method: **getDashboardWidgetList**

deleteDashboard `(String dashBoardId, UserInfo` userInfo`)`

This method is used to delete the dashboard specified by the dashboard ID passed as an argument.

Parameters:

param dashBoardId: dashboard Id of the Dashboard i.e. to be deleted

param userInfo: {@link com.intellica.client.common.UserInfo userInfo} object for authorization to get DashBoard.

```
dManager.deleteDashboard(dashBoardId, requestorUserInfo);
```

Get Dashboard Preferences

This Java API is used to get the list of the Dashboards set in its Preferences.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/DashBoard Management/GetDashboardPreferences.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a Dashboard Manager class object for Dashboard related settings.

```
DashboardManager dManager = new DashboardManager();
```

4. Get the Dashboard Preferences.

Method: **getDashboardPreferences**

Public ArrayList getDashboardPreferences`(String dashBoardId, UserInfo` userInfo`)`

Getting Dashboard Preferences Object.

Parameters:

param filterObj: Filter containing ONWER_APPID, OWNER_ORGID

param userInfo: {@link com.intellica.client.common.UserInfo userInfo} object for authorization to get DashBoard.

```
dbPreferencesList = dManager.getDashBoardPreferences(filterObj,
requestorUserInfo);
```

Schedules

Scheduling of reports is very helpful for better utilization of server and printer resources. Reports that take longer to run can be scheduled to save your time.

Report that needs processing of large volume of data and need server and printer resources for long time can be scheduled to be generated over the weekend when load on servers would be relatively low. By scheduling a report, it can be sent to multiple deliverables at a time, which is otherwise not possible.

Get the list of scheduled jobs

This Java API is used to get the list of All Scheduled Jobs.

This program:

1. Returns list of report schedules for a selected report.
2. Report schedule list can be filtered for ReportId and UserId

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/Schedules/ GetScheduleJobList.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a Scheduler Manager class object. This is the controller class for all operations related to scheduler. The class provides methods which acts as an interface for sending diffrent requests related to Scheduler to report engine from the jsps.

```
SchedulerManager schdMgr      =     new SchedulerManager();
```

4. Set the filter values corresponding to respective fields for getting filtered list of Scheduled Jobs.

```
// Create a Filter class Object
Filter filterObj = new Filter();

// set filter for private schedule jobs
filterObj.setFilterField(Enums.Filters.ScheduledJob.ISPUBLIC,
false);

// set filter for dedicated(Non-shared) schedule jobs
filterObj.setFilterField(Enums.Filters.ScheduledJob.ISSHARED,
false);

//Set the ReportId filter for getting all report schedules for
this Report
String reportId    = "93F21A40-01DD-DFFC-C874-A7E30A27A127";
filterObj.setFilterField(Enums.Filters.ScheduledJob.REPORTID,repo
rtId);

String userID      = "HostUser";
filterObj.setFilterField(Enums.Filters.ScheduledJob.USERID,userID
);
```

```
//Set the orgId filter for getting list of report schedules of a
user of this organization only
String orgId = "HostOrg";
filterObj.setFilterField(Enums.Filters.ScheduledJob.ORGID,orgId);
```

5. Get the Scheduled Job List based on filter applied.

Method: getScheduledJobList

```
public java.util.ArrayList getScheduledJobList(Filter filterObj,
UserInfo userInfo) throws SchedulerException
```

This method returns the scheduleJobList in ArrayList containing HashMap as each element for each row. The HashMap contains the details required for showing the list like the schedule job name, schedule job id etc. as name value pair.

Parameters:

- **Filter:** May take below fields

i.e.

```
Enums.Filters.ScheduledJob.ISPUBLIC
Enums.Filters.ScheduledJob.ISSHARED
Enums.Filters.ScheduledJob.REPORTID
Enums.Filters.ScheduledJob.ORGID
Enums.Filters.ScheduledJob.SCHEDULE
Enums.Filters.ScheduledJob.USERID
Enums.Filters.ScheduledJob.BATCHID
```

- **UserInfo:** The userInfo object for authorization.

```
// This will return array of Scheduled Jobs list from the
Intellicus Repository
ArrayList jobList = schdMgr.getScheduledJobList(filterObj,
                                    requestorUserInfo);
```

Create a Schedule Job

This Java API is used to create Scheduled Job that runs only once at a given time.

This program

1. Sets business parameters required to execute the report.
2. Creates a report delivery task.
3. Creates a ScheduleJob using 2).

Refer to

1. <Intellicus_Install_Path>/SampleCodes/Java APIs/Schedules/CreateDailyReportScheduleWithAllDeliveryOptions.java

2. <Intellicus_Install_Path>/SampleCodes/Java APIs/Schedules/CreateDailyReportScheduleWithEmailDelivery.java
3. <Intellicus_Install_Path>/SampleCodes/Java APIs/Schedules/CreateMonthlyReportSchedule.java
4. <Intellicus_Install_Path>/SampleCodes/Java APIs/Schedules/CreateNewSchedule.java
5. <Intellicus_Install_Path>/SampleCodes/Java APIs/Schedules/CreateOnceReportSchedule.java
6. <Intellicus_Install_Path>/SampleCodes/Java APIs/Schedules/CreateNewTask.java
7. <Intellicus_Install_Path>/SampleCodes/Java APIs/Schedules/DeleteScheduleJob.java
8. <Intellicus_Install_Path>/SampleCodes/Java APIs/Schedules/GetHistory.java
9. <Intellicus_Install_Path>/SampleCodes/Java APIs/Schedules/GetHistoryAndDetail.java
10. <Intellicus_Install_Path>/SampleCodes/Java APIs/Schedules/GetScheduleJobList.java
11. <Intellicus_Install_Path>/SampleCodes/Java APIs/Schedules/GetTaskParameters.java
12. <Intellicus_Install_Path>/SampleCodes/Java APIs/Schedules/RunNowScheduleJob.java
13. <Intellicus_Install_Path>/SampleCodes/Java APIs/Schedules/RunOnceScheduleJob.java
14. <Intellicus_Install_Path>/SampleCodes/Java APIs/Schedules/ScheduleReportlist.java

for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a SchedulerManager class object. This is the controller class for all operations related to scheduler. The class provides methods acting as an interface for sending diffrent requests related to Scheduler to report engine from the jsps.

```
SchedulerManager schdMgr      =     new SchedulerManager();
```

4. Create an instance of Task and set its various properties.

```
Task task          =     new Task();
String batch_Name = "Task_Once";
task.setBatchName(batch_Name);
String rep_Id = "93F21A40-01DD-DFFC-C874-A7E30A27A127";
task.setReportID(rep_Id);
task.setReportFormat(InteraConstants.ReportFormats.PDF);
task.setIsPublic(false);
task.setIsShared(false);
```

5. Set the Delivery Options for Task and related properties.

For E-mail

```
task.setDeliveryOperationType(com.intellica.client.common.Enums.B
atch.DispatchOperationTypes.EMAIL);

// get EmailProperty object for setting Email attributes
EMailProperty  eMailProp     =     task.getEMailProperty();

String mail_To = "HostUser@HostOrg.com";
String mail_Sub = "Mail Subject goes here";
String  mailMsg  =  "Dear  User,  Please  find    <%Report_Name%>
attached";

eMailProp.setMailAttach(true);
eMailProp.setMailTO(mail_To);
eMailProp.setMailSUB(mail_Sub);
eMailProp.setMailMSG(mailMsg);
```

For FTP

Part of Sample Code implementing FTP Delivery Option**:**

```
task.setDeliveryOperationType(com.intellica.client.common.Enums.B
atch.DispatchOperationTypes.FTP);

String serverName = "ftp.intellicus.com";
String username = "upload";
String password = "download";
String folderName = "transferbin";
String fileName = "MyFile";

// get FTPProperty object for setting Email attributes
FTPProperty ftpProp = task.getFTPProperty();
ftpProp.setServerName(serverName);
ftpProp.setUserName(username);
ftpProp.setPassword(password);
ftpProp.setFolderName(folderName);
ftpProp.setFileName(fileName);
```

For Publish Report

Part of Sample Code implementing Publish Delivery Option:

```
task.setDeliveryOperationType(com.intellica.client.common.Enums.B
atch.DispatchOperationTypes.PUBLISH);
```

```
//get PublishProperty object for setting publish attributes for
//the task
PublishProperty publishProp = task.getPublishProperty();

String fileName = "Shared_Publish1";
//setter method for saveFileName member variable.
publishProp.setSaveFileName(fileName);

//setter method for isPublic member variable.
publishProp.setIsPublic(true);     //True : Set the Published
                                   //Report as Public

//setter method for periodType member variable.
//Enums.Batch.PeriodType.ENDPERIOD : valid upto end of Day, Hour,
//Month, Week, Year.
//For ENDPERIOD, call setEndPeriod to set the value to Day, Hour,
//Month, Week, Year.
//Enums.Batch.PeriodType.EXPIRYDATE  :  valid  upto  a  particular
//date
//For  EXPIRYDATE,  call  setPublishValidUptoFixedDate  to  set  the
//end Date.
//Enums.Batch.PeriodType.INTERVALPERIOD   :   valid   upto   fixed
//interval
//For  INTERVALPERIOD,  call  setIntervalValue  and  then  set  the
//value for Interval Period.
publishProp.setPeriodType(Enums.Batch.PeriodType.ENDPERIOD);

//setter method for endPeriod member variable when Period Type is
//set to ENDPERIOD
//It sets the endPeriod value to either Hour/Day/Week/MonthYear.
publishProp.setEndPeriod(com.intellica.client.common.Enums.Batch.
EndPeriod.YEAR);
```

For Print Report

Part of Sample Code implementing Publish Delivery Option:

```
task.setDeliveryOperationType(com.intellica.client.common.Enums.B
atch.DispatchOperationTypes.PRINT);
//This class holds PrintProperty for Task.
//All the variables are used for making a print request.
PrintProperty printProp = task.getPrintProperty();

int printCopies = 1;
```

```
String printerName = "Microsoft XPS Document Writer";

//Number of copies to be printed
printProp.setPrintCopies(1);
//Printer name that is used for printing
printProp.setPrinterName(printerName);
//Set the number of pages to be printed
printProp.setPrintPageRange("All");
```

6. Create Scdedule for the Scheduled Job and set its Frequency Type

- Daily Freequency Type

To Set Frequency type as **Daily**
Enums.Schedule.FrequencyTypes.DAILY

```
Schedule schedule  =     new Schedule();
schedule.setFrequencyType(Enums.Schedule.FrequencyTypes.DAILY);
```

- Monthly Freequency Type

To Set Frequency type as **Monthly**
Enums.Schedule.FrequencyTypes.MONTHLY

```
Schedule schedule   =     new Schedule();
schedule.setFrequencyType(Enums.Schedule.FrequencyTypes.MONTHLY);
```

- Weekly Freequency Type

To Set Frequency type as **Weekly**
Enums.Schedule.FrequencyTypes.WEEKLY

```
Schedule schedule  =     new Schedule();
schedule.setFrequencyType(Enums.Schedule.FrequencyTypes.WEEKLY);
```

7. Create Scheduled Job and associate created Task and Schedule to this Scheduled Job.

```
//create a SCHEDULEDJOB
ScheduleJob  schdJob    =     new ScheduleJob(task,schedule);
```

8. Set Run Type for Scheduled Job

This is setter method for runType member variable.

public void setRunType(java.lang.String runType)

```
Enums.ScheduledJob.JobTypes.NOW
```

```
Enums.ScheduledJob.JobTypes. ONCE
Enums.ScheduledJob.JobTypes.RECUR
```

Parameters:

- **RunType: T**akes String value.

Part of Sample code for "NOW" as Run-type:

```
ScheduleJob  schdJob          =     new ScheduleJob();
schdJob.setRunType(Enums.ScheduledJob.JobTypes.NOW);
```

Part of Sample code for "ONCE" as Run-type:

```
ScheduleJob  schdJob          =     new ScheduleJob();
schdJob.setRunType(Enums.ScheduledJob.JobTypes.ONCE);
schdJob.setOnceDate("10/03/2007");
schdJob.setOnceTime("10:52:10");
```

Part of Sample code for "RECUR" as Run-type:

```
ScheduleJob  schdJob          =     new ScheduleJob();
schdJob.setRunType(Enums.ScheduledJob.JobTypes.RECUR);
schdJob.setFrequencyType(Enums.Schedule.FrequencyTypes.DAILY);
schdJob.setAfterDays(2);
```

9. Add the Scheduled Job in the Repository.

Method : addScheduleJob(schdJob,userInfo)
public void addScheduleJob(ScheduleJob scheduleJob, UserInfo userInfo)
throws SchedulerException
Adds ScheduleJob alongwith Schedule and Task depending on whether they are created Shared or Dedicated in the Intellicus Repository.

Parameters:

- **ScheduleJob:** The ScheduleJob object.
- **UserInfo:** The userInfo object for authorization.

```
schdMgr.addScheduleJob(schdJob,requestorUserInfo);
```

Delete Schedule Job

This Java API is used to delete dedicated scheduledJob selected from the list.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/Schedules/DeleteScheduleJob.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a SchedulerManager class object. This is the controller class for all operations related to scheduler. The class provides methods which acts as an interface for sending diffrent requests related to Scheduler to report engine from the jsps.

```
SchedulerManager schdMgr      =     new SchedulerManager();
```

4. Delete Scheduled Job with given scheduledJob Id.

Method : deleteScheduledJob

```
public void deleteScheduledJob(java.lang.String jobID,
                               UserInfo userInfo)
                        throws SchedulerException
```

This method deletes an existing ScheduledJob.

Parameters:

- **JobID:** Takes a String value.
- **UserInfo:** The userInfo object for authorization.

```
String schdJobId   =     "114016239210";
schdMgr.deleteScheduledJob(schdJobId,requestorUserInfo);
```

Get the input parameter names and their values set at the task creation time.

This Java API is used to get the input Parameter names whose values are set at the Task Creation time.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/Schedules/GetTaskParameters.java for sample code of this use case.

This program:

1. Prepare a scheduler manager object
2. Get the task object for the specified task id
3. Get the hash map of input parameter from the task object
4. Get the input parameter names and their values set at the task creation time.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a SchedulerManager class object. This is the controller class for all operations related to scheduler. The class provides methods acting as an

interface for sending different requests related to Scheduler to report engine from the jsps.

```
SchedulerManager schdMgr      =     new SchedulerManager();
```

4. Get the Task instance for given Task Id whose input parameters are to be obtained.

```
String taskId      =     "1169627272589";
Task taskObj =     schdMgr.getTask(taskId, requestorUserInfo);
```

5. Get the Input Parameters for this Task.

Method: getInputParamMap

```
public java.util.HashMap getInputParamMap()
```

getter method for inputParamMap member variable.

Returns:
inputParamMap as HashMap.

```
HashMap inputParamMap    =     taskObj.getInputParamMap();
```

Cab Deployment

Upload and Deploy cab/irb file

This Java API is used to upload and deploy cab/irb file both.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/Cab Deployer/CabDeployer.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Set the application path and cab/irb File path.

```
//path where the intellicus web client is deployed
String           applicationPath           =           "C:/Program
Files/Intellicus/Jakarta/webapps/intellicus";

CabFile.setApplicationPath(applicationPath+ File.pathSeparator);

//actual path of irb file
String irbFilePath = "c:/ABC.irb";
```

4. Upload the Cab File.

```
CabFile  cabFileObj  = new
CabFile(irbFilePath,requestorUserInfo);
```

5. Deploy the Cab File.

Method: deploy(java.lang.String cabPath, UserInfo userInfo)

```
public java.util.HashMap deploy(java.lang.String cabPath,
                                 UserInfo userInfo)
```

This method takes the path of the cab file i.e. to be deployed and deploys it in the Intellicus Repository.

Part of Sample Code implementing "deploy":

```
cabFileObj.deploy(irbFilePath,irbFilePath+".log"
,requestorUserInfo);
```

Report Object

Query Object

Query objects contain SQL query and list of the fields along with their attributes for using in Ad hoc reports and sometimes in Standard reports.

These are designed from Intellicus web portal at Main Menu -> Repository-> Report Objects -> Query Objects screen.

These objects are stored as XML in Intellicus repository.

The Intellicus web UI provides option for Adding, Deleting and Editing Query Objects.

Almost all of the operations provided by the Intellicus web UI are also available from the Java APIs.

This document intends to explain the Java APIs with Query Object manipulation purpose; there would be sample code files associated with this document.

Attribute of Query Objects:

Below table lists all the attributes of query objects which can be modified through Java APIs.

Information	Type	Description
Name	Mandatory	Must be Unique
SQL Statement	Mandatory	The SQL Statement which sources data for this QO.
Connection Name	Optional	If provided, then this Query object becomes associated to the given connection.

Information	Type	Description
Column Details	Optional As array list.	These are the details of the fields.
Column field Name	Mandatory (if column details are given)	Must be returned by above SQL. Here it is used to match the field and associate below given caption and width attributes to it.
Column data type	Mandatory, CHAR/NUMBER/DATE	Default – Field data type returned by SQL.
Column Caption	Optional, String	Default – Field name returned by SQL with spaces truncated and Case converted to Title case.
Column Width	Mandatory, Integer	
Column Hidden	Optional, Boolean	Default – False
Column Hyper Link	Optional, String	
Column Key Field Name	Optional, String	
Column Group Label	Optional, String	Adds this column to a group, Group label must exist.
Column Alignment	Optional	Default- Left for Char and Date, Right for Numeric

Add Report Object

This Java API is used for creating a new Report Object.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/ReportObjects/ AddReportObject.java for sample code of this use case.

In this API, you would require to provide SQL query to create the Query Object.

You may also provide column details like column name, caption, data type, width, output format and hyperlink.

Refer to attributes table for the list of attributes you can set during creation.

Steps

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create query object by using the following method:

Method

```
Enums.IRO.TYPE.QUERY.QUERY
Enums.IRO.TYPE.QUERY.PARAMETER
Enums.IRO.TYPE.QUERY.nFormat
```

```
Enums.IRO.TYPE.QUERY.Format
Enums.IRO.TYPE.QUERY.nParameter
Enums.IRO.TYPE.QUERY.nQuery
Enums.IRO.TYPE.QUERY.nReportObjectType
```

```
public static ReportObject createReportObject
             (Enums.IRO.TYPE.QUERY) of ReportObject class.
```

4. **Set the SQL Query** - SQL can be set by creating com.impetus.intera.layout.sqleditor.SQLEditor class object and calling setSQL () method.

```
SQLEditor sqlEditor = qoObj.getSqlEditor();
  sqlEditor.setSQL(sqlQuery,true);
```

Parameters:

- **sqlQuery:** This is to specify the query string to be used for fetching data.

5. **Set the columns**- Columns can be set by passing ArrayList containing the QueryObjectColumn object. (QueryObjectColumn contains column attributes column name, data type caption, and width and others as mentioned in above table).

```
Public void setColumnDetails (java.util.ArrayList columnDetails)
```

Parameters:

- **ColumnDetails:** This is the array list of columns to be added in the query object.

Objects of QueryObjectColumn can be created by using the constructor of com.impetus.intera.reportobjects.QueryObjectColumn in which user provides attributes like column name, caption, width and datatype to the columns.

```
QueryObjectColumn(java.lang.String columnName,
                  java.lang.String caption,
                  java.lang.String width,
                  java.lang.String dataType)
```

```
String colName = "Product";
String caption = "Product Name";
QueryObjectColumn qoc1 = new QueryObjectColumn(colName, caption
,"","");
```

6. Optionally, You may set other attributes of this column also.

```
// To hide this column
qoc1.setHidden(true);

String groupCaption = "Job";
//To set this column in a group
qoc1.setGroupLabelCaption(groupCaption);

// To set hyperlink for this column.
String hyperLink = "http://www.google.com";
qoc1.setHyperlink(hyperLink);
```

7. Set the connection name- setConnectionName method of com.impetus.intera.reportobjects.QueryObject class can be used to set the name of the connection used by Query Object

```
public void setConnectionName(java.lang.String connectionName)
```

Parameters:

- **ConnectionName:** The database connection Name to be used for query object.

5. Set if filter is mandatory-This method is used to set the isFilterMandatory.

```
public void setIsFilterMandatory(boolean isFilterMandatory)
of QueryObjects class.
```

Parameters:

IsFilterMandatory: Whether mandatory filters are applied or not.

6. Set the group caption.

```
public void setGroupCaption(boolean isGroupCaption)
of QueryObjects class.
```

8. Add the query object. This can be done by calling addReportObject method of ReportObjectManager.]

```
public void addReportObject(ReportObject reportObject,
                            UserInfo userInfo).
```

Parameters:

- **ReportObject:** Java object of Query object to be deleted.

- **UserInfo:** Object of UserInfo class.

Part of Sample Code implementing "Add Report Object"

```
ReportObjectManager reportObjectManager = new
ReportObjectManager();

String qoName = "TestQuery";
String queryId = "qoObjID";
String connName = "ReportDB";

qoObj.setName(qoName);
qoObj.setId(queryId);
qoObj.setCached(true);
qoObj.setConnectionType("Default");
qoObj.setConnectionName(connName);
qoObj.setSource("SQL");

reportObjectManager.addReportObject(qoObj, requestorUserInfo);
```

Add Query Object from CSV Source

This Java API is used for creating a new Query Object from a CSV source file.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/ReportObjects/ AddReportObject.java for sample code of this use case.

In this API, you would require to provide SQL query to create the Query Object.

You may also provide column details like column name, caption, data type, width, output format and hyperlink.

Refer to attributes table for the list of attributes you can set during creation.

Steps

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create CSV Source Object

```
//CSVSource object
CSVSource csvSource =new CSVSource();
```

4. Set path for the CSV Source

```
//Set path of file along with name under connection
//if connection is created at "E:\csv" and CSV file is in
//"E:\csv\subfolder\test.csv"
```

```
//then setpath will be csvSource.setPath("subfolder/test.csv");
csvSource.setPath("test/deptdetails_test.csv");
//Set separator if not a default separator i.e ','
csvSource.setSeparator("|");
```

5. Create query object with this CSV Source and set various attributes

```
QueryObject qoObj=QueryObject.createQueryObject("QoName01",
csvSource, "ConnectionFile", requestorUserInfo);

//To set the ID for Query Object manually
qoObj.setId("QoId01");

//Set Category ID in order to create query object in a specific
category.
qoObj.setCategoryId("Cat1");

//To set caching true or false for the Query Object
qoObj.setCached(true);

//To set the description for the Query Object
qoObj.setDescription("This is a Test Query .");
```

6. Save query Object in Repository

```
//Save Query Object
qoObj.save(Enums.QueryObject.Action.ADD, requestorUserInfo);
```

Get Query Object by Name

This Java API is used to get a java object representing Query Object by its name.

The object can be used to fetch details and/or then to modify details and replace Query Object in the repository.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/ReportObjects/ GetReportObjectByName.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Call getReportObjectByName method of com.impetus.intera.reportobjects.ReportObjectManager by type casting returned object to com.impetus.intera.reportobjects.QueryObject

```
public ReportObject getReportObjectByName
           ( String reportObjectType,
             String reportObjectName,
```

```
                UserInfo userInfo)
```

Parameters

- **Report Object Type:** Pass *Enums.IRO.TYPE.QUERY.*
- **Report Object Name:** Name of the Query Object to be fetched.
- **User Info:** Credentials of user requesting this information.

For getting attributes of Query Object you need to type-cast the Report Object into Query Object.

```
QueryObject
qoObj=(QueryObject)reportObjectManager.getReportObjectByName(Enum
s.IRO.TYPE.QUERY,qoName,userInfo);
```

4. Call getColumnDetails method on the java object of query object returned. This method will return array list of all the columns.

Method

```
public java.util.ArrayList getColumnDetails()
```

Returns:

Returns the array list of columnDetails.

5. Call getConnectionName method on the java object of query object returned. This method is used to get the name of the connection used by Query Object

```
public java.lang.String getConnectionName()
```

Returns:

Returns the name of the connection.

Call isFilterMandatory method on the java object of query object returned.

```
public boolean isFilterMandatory()
```

Returns:

Returns whether mandatory filters are applied or not.

Part of Sample Code implementing "get Report Object By Name"

```
ReportObjectManager        reportObjectManager        =        new
ReportObjectManager();

String qoName = "All Country Sales";
```

```
QueryObject
qoObj=(QueryObject)reportObjectManager.getReportObjectByName(Enum
s.IRO.TYPE.QUERY,qoName, requestorUserInfo);
```

Get Report Object List

This Java API is used to retrieve the list of Report Objects from Intellicus repository.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/ReportObjects/ GetReportObjectList.java for sample code of this use case.

Steps

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Call getReportObjectList of com.impetus.intera.reportobjects.ReportObjectManager to retrieve an array List of Query Objects from the Intellicus repository.

```
public java.util.ArrayList getReportObjectList(java.lang.String
roType,UserInfo userInfo)
```

Parameters:

- **Report Object Type:** Pass Enums.IRO.TYPE.QUERY.

The returned java.util.ArrayList will have each element as a Query Object retrieved from the Intellicus repository.

Part of Sample Code implementing "get Query Object List"

```
ReportObjectManager reportObjectManager = new
ReportObjectManager();

ArrayList reportObjList=new ArrayList();

ReportObject
repObjectQuery=ReportObject.createReportObject(Enums.IRO.TYPE.QUE
RY);

reportObjList=reportObjectManager.getReportObjectList(repObjectQu
ery.getType(),requestorUserInfo);
```

Other related APIs are:

```
public ArrayList getReportObjectList(String roType, Filter
filterObj, UserInfo userInfo)
```

```
public ArrayList getReportObjects(String roType, boolean cached,
UserInfo userInfo)
```

```
public ArrayList getReportObjects(String roType, UserInfo
userInfo)
```

Get Parameter Object List

This Java API is used to retrieve the list of Query Objects from Intellicus repository.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/ReportObjects/ GetReportObjectList.java for sample code of this use case.

Steps

1. Call getReportObjectList of com.impetus.intera.reportobjects.ReportObjectManager to retrieve an array List of Query Objects from the Intellicus repository.

```
public java.util.ArrayList getReportObjectList(java.lang.String
roType,UserInfo userInfo)
```

Parameters:

- **Report Object Type:** Pass Enums.IRO.TYPE.QUERY.

The returned java.util.ArrayList will have each element as a Query Object retrieved from the Intellicus repository.

Part of Sample Code implementing "get Parameter Object List".

```
ReportObjectManager reportObjectManager = new
ReportObjectManager();

ArrayList reportObjList=new ArrayList();

ReportObject
repObjectQuery=ReportObject.createReportObject(Enums.IRO.TYPE.
PARAMETER);

reportObjList=reportObjectManager.getReportObjectList(repObjectQu
ery.getType(),requestorUserInfo);
```

Delete Report Object

This Java API is to delete an existing Report Object from Intellicus Repository.

For deleting a report object, you would require to provide the name of the Query Object to be deleted. If report object with the given name exists, then that would be deleted, otherwise it gives an error message that "No such Query Object Found".

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/ReportObjects/ DeleteReportObject.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Get Report Object –Retrieve the Report Object to be deleted from the Intellicus Report repository. For this, call getReportObjectByName() method of com.impetus.intera.reportobjects.ReportObjectManager.

```
Public ReportObject getReportObjectByName (Enums.IRO.TYPE.QUERY,
String reportObjectName, UserInfo userInfo)
```

Parameter:

- **Report Object type:** Pass Enums.IRO.TYPE.QUERY for the Query Object.
- **ReportObjectName:** The name of the query object to be deleted.
- **UserInfo:** Object of UserInfo class.

```
//To get the Object for the Query Object Class.
QueryObject qoObj=(QueryObject)reportObjectManager.
getReportObjectByName(Enums.IRO.TYPE.QUERY,qoName,userInfo);
```

Delete the Report Object – For deleting the Report Object, call deleteReportObject method of

```
com.impetus.intera.reportobjects.ReportObjectManager

public void deleteReportObject(ReportObject reportObject,
                               UserInfo userInfo)
```

Parameters:

- **ReportObject:** Java object of Query object to be deleted.
- **UserInfo:** Object of UserInfo class.

```
  //This will used to Delete the Query Object.
reportObjectManager.deleteReportObject(qoObj,userInfo);
```

Part of Sample Code implementing "Delete Report Object"

```
ReportObjectManager reportObjectManager = new
ReportObjectManager();

String qoName = "newqo1";
QueryObject
qoObj=(QueryObject)reportObjectManager.getReportObjectByName(Enum
s.IRO.TYPE.QUERY,qoName,userInfo);
reportObjectManager.deleteReportObject(qoObj,requestorUserInfo);
```

Replace Query Object

Query object can be replaced by calling the replaceReportObject() method of com.impetus.intera.reportobjects.ReportObjectManager.

```
  reportObjectManager.replaceReportObject(qObject,userInfo);
```

Part of Sample Code implementing "Replace Report Object"

```
ReportObjectManager        reportObjectManager        =        new
ReportObjectManager();

String qOName="TestReplace";

QueryObject
qObject=(QueryObject)reportObjectManager.getReportObjectByName(En
ums.IRO.TYPE.QUERY,qOName,requestorUserInfo);

String colName = "BANKS.BANKID";
String caption = "BankID";
String width = "3";
String dataType = "NUMBER";
String grpLabelCaption = "Job";
//create the column to be added and set its attributes.
QueryObjectColumn             qocAdd             =             new
QueryObjectColumn("BANKS.BANKID","BankID","3","NUMBER");
qocAdd.setAlignment(1);
qocAdd.setHidden(false);
qocAdd.setGroupLabelCaption(grpLabelCaption);

//add this column in the query object
qObject.addColumn(qocAdd);
```

```
reportObjectManager.replaceReportObject(qObject,requestorUserInfo
);
```

Other related APIs for accessing QO/PO are:

1. public LinkedHashMap getAllPOMap()
2. public LinkedHashMap getAllQOMap()

OLAP

Import

```
//OLAP related imports
import com.impetus.intera.layout.datasource.qo.QODataSource;
import com.impetus.intera.layout.upx.FetchDataSource;
import com.impetus.intera.reportobjects.CubeObject;
import com.impetus.intera.reportobjects.QueryObjectColumn;
import com.impetus.intera.reportobjects.QueryObjectFacade;
import com.impetus.intera.reportobjects.ReportObjectException;
import com.impetus.intera.reportobjects.ReportObjectManager;
import com.intellica.client.common.EntityProperties;
import com.intellica.client.common.EntityProperty;
import com.intellica.client.common.Enums;
import com.intellica.client.layout.LayoutManager;
import com.intellica.client.olap.CubeDataSource;
import com.intellica.client.olap.DimMapping;
import com.intellica.client.olap.Dimension;
import com.intellica.client.olap.Hierarchy;
import com.intellica.client.olap.Level;
import com.intellica.client.olap.Measure;
import com.intellica.client.olap.MeasureGroups;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
```

Cube Object

AddCube

This Java API is used to create a new Cube Object in the Intellicus Repository.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/OLAP/ AddCube.java for sample code of this use case.

Steps :

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. addReportObject() method of ReportObjectManager class

Method: addReportObject

This is the method of ReportObjectManager class i.e. used to add Cube Object in Repository

```
public void addReportObject(ReportObject reportObject,
                            UserInfo userInfo).
```

Parameters:

- **ReportObject:** Java object of Cube object to be added.
- **UserInfo:** Object of UserInfo class.

Code Snippet of the sample code-

Set the Query Id for which Cube Object is to be created.

```
String queryId = "SampleCubeQuery";
```

Set other details.

```
String categoryId = "CT"; //Category in which cube object is to
be added
String measureName = "Count";
String measureDataField = "ResourceId";
String measureSummaryFunction = "2"; //1 for sum, 2 for count
//Set cube object properties
CubeObject cubeObj = new CubeObject();
cubeObj.setCategoryId(categoryId);   //category   Id   where   Cube
Object is to be saved.
cubeObj.setName("Cube "+queryId);
cubeObj.setId("Cube "+queryId);
cubeObj.setCoType("INTERNAL");
cubeObj.setCubeUniqueName("Cube "+queryId);

FetchDataSource fetchDataSource = new FetchDataSource();

QODataSource qoDataSrc = new QODataSource();
qoDataSrc.setId(queryId);   //Query Object Id
fetchDataSource.setSource("QO");
fetchDataSource.setSourceObj(qoDataSrc);
```

```
HashMap<String, FetchDataSource> cubeQueries = new
HashMap<String, FetchDataSource>();
cubeQueries.put("CUBE_QUERY_"+queryId, fetchDataSource);
cubeObj.setCubeQueries(cubeQueries);
```

Adding Dimensions

For adding dimensions, first iterate over the list of columns/fields of Query object, then add them.

```
ArrayList<Dimension> dimensionList = new ArrayList<Dimension>();
//Getting columns/fields in QO so as to iterate to create
Dimensions
QueryObjectFacade queryObjFacade =
(QueryObjectFacade)rom.getReportObjectFacadeById(Enums.IRO.TYPE.Q
UERY, queryId, requestorUserInfo);
ArrayList<QueryObjectColumn> qoColumnsList =
queryObjFacade.getColumnList();
Iterator itr = qoColumnsList.iterator();

//iterating through the Column List
while(itr.hasNext()){
  QueryObjectColumn qoColumn  = (QueryObjectColumn)itr.next();
  System.out.println("Column Name = "+qoColumn.getColumnName());
  columnName = qoColumn.getColumnName();

  dimMappingObj = new DimMapping();
  dimMappingObj.setDimensionId("DIMENSION_"+columnName);
  dimMappingMap.put("DIMENSION_"+columnName,dimMappingObj);

  Dimension dimensionObj = new Dimension();
  dimensionObj.setDimID("DIMENSION_"+columnName);
  dimensionObj.setIsValuesRestricted(false);
  dimensionObj.setGISEnabled(false);
  dimensionObj.setUniqueName("DIMENSION_"+columnName);
  dimensionObj.setName(columnName);
  dimensionObj.setType("REGULAR");
  HashMap<String, CubeDataSource> dataSourceObj = new
HashMap<String, CubeDataSource>();
  CubeDataSource cubeDataSourceObj = new CubeDataSource();
  cubeDataSourceObj.setId("CUBE QUERY "+queryId);
  cubeDataSourceObj.setType("REFERENCED");
  dataSourceObj.put("CUBE_QUERY_"+queryId,cubeDataSourceObj );
  dimensionObj.setFetchSourcesForDimension(dataSourceObj);

  ArrayList<Hierarchy> hierarchyList = new
ArrayList<Hierarchy>();
```

```
  Hierarchy hierachy = new Hierarchy();
  hierachy.setName("HIERARCHY");
  hierachy.setUniqueName("DIMENSION_HIERARCHY_"+columnName);
  ArrayList<Level> levelList = new ArrayList<Level>();
  Level level = new Level();
  //to allow All as member value of a dimension
  if(allowAll){
        level.setAll(true);
        level.setUniqueName("DIEMENSION_LEVEL_0");
        level.setName("Hierarchy.ALL");
        levelList.add(level);
  }
  level = new Level();
  level.setAll(false);
  level.setUniqueName("DIEMENSION_LEVEL_1");
  level.setName(columnName);
  level.setDataField(columnName);
  level.setAssociatedDimQueryId("CUBE_QUERY_"+queryId);
  levelList.add(level);

  hierachy.setLevels(levelList);
  hierarchyList.add(hierachy);
  dimensionObj.setHierarchies(hierarchyList);
  dimensionList.add(dimensionObj);
}

cubeObj.setDimMappings(dimMappingMap);
```

Adding the measure Dimension

```
//Adding Measure dimension
Dimension dimensionObj = new Dimension();
dimensionObj.setDimID("Dim_Measures");
dimensionObj.setIsValuesRestricted(false);
dimensionObj.setGISEnabled(false);
dimensionObj.setUniqueName("Measures");
dimensionObj.setName("Measures");
dimensionObj.setType("MEASURE");

ArrayList<Hierarchy> hierarchyList = new ArrayList<Hierarchy>();
Hierarchy hierachy = new Hierarchy();
hierachy.setUniqueName("Measures");
ArrayList<Level> levelList = new ArrayList<Level>();
Level level = new Level();
```

```
level.setAll(false);
level.setDataType("Regular");
level.setUniqueName("MeasuresLevel");
levelList.add(level);

hierachy.setLevels(levelList);
hierarchyList.add(hierachy);
dimensionObj.setHierarchies(hierarchyList);
dimensionList.add(dimensionObj);

cubeObj.setDimensions(dimensionList);

String measureId = "MEASURE_1";
MeasureGroups measureGrpsObj = new MeasureGroups();
Map<String, ArrayList<String>> measureMap = new HashMap<String,
ArrayList<String>>();
ArrayList<String> measureIdList = new ArrayList<String>();
measureIdList.add(measureId);
measureMap.put("MeasureGroup", measureIdList);
measureGrpsObj.setMeasureGroups(measureMap);
measureGrpsObj.setMeasureGroups(measureMap);

Map<String, CubeDataSource> dataSourceMap = new HashMap<String,
CubeDataSource>();
ArrayList<String> DataSourceId = new ArrayList<String>();
CubeDataSource cubeDataSrcObj = new CubeDataSource();
cubeDataSrcObj.setId("CUBE_QUERY_"+queryId);
cubeDataSrcObj.setType("REFERENCED");
dataSourceMap.put("CUBE_QUERY_"+queryId, cubeDataSrcObj);
measureGrpsObj.setFetchSources(dataSourceMap);

cubeObj.setMeasureGroups(measureGrpsObj);

//Adding Measures
HashMap<String,Measure>          measureList          =          new
HashMap<String,Measure>();
Measure measureObj = new Measure();

measureObj.setId(measureId);
measureObj.setUniqueName(measureId);
measureObj.setName(measureName);
measureObj.setIsDefault(true);
//measureObj.setFormatType("2");
measureObj.setFormat("0");
measureObj.setQueryID("CUBE_QUERY_"+queryId);
```

```
measureObj.setDataField(measureDataField);
measureObj.setSummaryFunction(measureSummaryFunction);
measureList.put(measureId, measureObj);
cubeObj.setMeasures(measureList);
```

Calling addReportObject() method

```
//Method used to add Cube Object in Repository
rom.addReportObject(cubeObj, requestorUserInfo);
```

Get Dimensions List

This Java API is used to get the list of dimensions for provided Cube Object Id.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/OLAP/GetDimensionList.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. addReportObject() method of ReportObjectManager class

Method: getDimensions

This is the method of CubeObject class i.e. used to get the dimensions of requested Cube object

```
public ArrayList<Dimension> getDimensions()
```

Code Snippet of the sample code-

Set the Cube Object Id whose dimensions are to be fetched.

```
String cubeName = "SampleCubeQuery";
```

Getting Cube Object Detail to get its dimensions.

```
OLAPManager olapMgr = new OLAPManager();
CubeObject cubeObj =
(CubeObject)olapMgr.getCubeObjectDetail(cubeName, cubeName,
requestorUserInfo);
ArrayList<Dimension> dimList = cubeObj.getDimensions();
Dimension dim = null;
//iterating through the list of dimensions
for(int i=0;i<dimList.size();i++){
      dim = (Dimension)dimList.get(i);
```

```
      System.out.println("Dimension Id = "+dim.getUniqueName()+",
Name = "+dim.getName());
}
```

Delete Cube Object

This Java API is used to delete the cube object from the Intellicus Repository for given Cube Object Id.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/OLAP/DeleteCubeObject.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Delete the Report Object

Method: deleteReportObject
For deleting the Report Object, call deleteReportObject method of ReportObjectManager

```
com.impetus.intera.reportobjects.ReportObjectManager

public void deleteReportObject(ReportObject reportObject,
                               UserInfo userInfo)
```

Parameters:

- **ReportObject:** Java object of Cube object to be deleted.
- **UserInfo:** Object of UserInfo class.

```
//This will be used to delete the Cube Object.
reportObjectManager.deleteReportObject(qoObj,userInfo);
```

Part of Sample Code implementing "Delete Cube Object"

Set the Cube Object Id that is to be deleted.

```
//Cube object Id that is to be deleted
String cubeObjId = "Cube_QOForCube_1";
```

Getting the object of Cube and deleting it from Repository

```
//getting ReportObject corresponding to provided Cube Object Id
```

```
ReportObject rObj = (ReportObject)rom.getReportObject
(Enums.IRO.TYPE.CUBEOBJECT, cubeObjId, cubeObjId, true
,requestorUserInfo);

//Method to delete Cube
rom.deleteReportObject(rObj, requestorUserInfo);
```

Build Cube Object

This Java API is used to build the cube for provided Cube Object.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/OLAP/BuildCubeObject.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Build the Cube Object

Method: buidCubeObject
For building the cube, call method of ReportObjectManager

```
com.impetus.intera.reportobjects.ReportObjectManager

public void buildCubeObject(String cubeBuildXML, UserInfo
userInfo) throws ReportObjectException{
```

Parameters:

- **cubeBuildXML:** XML of CubeObject Build.
- **UserInfo:** Object of UserInfo class.

Part of Sample Code implementing "Build Cube Object"

Set the Cube Object Id that is to be build.

```
//Cube object Id that is to be deleted
String cubeObjId = "Cube_QOForCube_1";
```

Getting the object of Cube and deleting it from Repository

```
String buildXML = "<CUBEOBJECT_BUILD ID=\""+cubeObjId+"\">\n";
buildXML += "\t\t\t<BUILD_INFO>\n\t\t\t\t<PROPERTIES>\n";
buildXML += "\t\t\t\t<PARAM NAME=\"BUILD_ON_HADOOP\">\n";
buildXML += "\t\t\t\t\t<VALUE><![CDATA[FALSE]]></VALUE>\n";
```

```
buildXML += "\t\t\t\t</PARAM>\n";
buildXML += "\t\t\t\t</PROPERTIES>\n";
buildXML += "\t\t\t</BUILD_INFO>\n";
buildXML += "<SCHEDULE_INFO
JOBTYPE=\"NOW\"></SCHEDULE_INFO></CUBEOBJECT_BUILD>";

//method used to build the cube object.
rom.buildCubeObject(buildXML, requestorUserInfo);
```

Get Build Status

This Java API is used to get the build status for the given Cube.
Posiible values of build status –
"BUILDING", "COMPLETED", "PENDING", "FAILED", "BUILD_SUBMITTED"

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/OLAP/GetBuildStatus.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Get the build status for the Cube Object

Method: getCubeObjectBuildStatus
For getting the build status, call method of ReportObjectManager

```
com.impetus.intera.reportobjects.ReportObjectManager

public COBuildStatus getCubeObjectBuildStatus(Filters
filters,UserInfo userInfo) throws ReportObjectException{
```

Parameters:

- **filters** : filters to get CubeObject Build Status
- **UserInfo:** Object of UserInfo class.

Part of Sample Code implementing "Get Build Status"

Set the Cube Object Id that is to be build.

```
//Cube object Id whose build status is to get
String cubeObjId = "SalesCube";
```

Getting the object of Cube and deleting it from Repository

```
Filters filters = new Filters();
```

```
Filter filter = new Filter("CUBEOBJECT_ID", "", cubeObjId);
filters.add(filter);
COBuildStatus coBuildStatus =
rom.getCubeObjectBuildStatus(filters, requestorUserInfo);
System.out.println("Build Status = "+coBuildStatus.getStatus()
```

Cancel Build

This Java API is used cancel the build for which build cube request is sent to Report Engine

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/OLAP/CancelBuild.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Get the build status for the Cube Object

Method: cancelCubeObjectBuild
For cancelling the cube build process, call method of ReportObjectManager

```
com.impetus.intera.reportobjects.ReportObjectManager

public void cancelCubeObjectBuild(HashMap requestParams,UserInfo
userInfo) throws ReportObjectException{
```

Parameters:

- **requestParams :** Request Params required to cancel Cube Object Build(REQUEST_ID)
- **UserInfo:** Object of UserInfo class.

Part of Sample Code implementing "Cancel Build"

Set the Cube Object Id that is to be build.

```
//Cube object Id whose build status is to get
String cubeObjId = "SalesDataCube";
```

Getting the object of Cube and deleting it from Repository

```
Filters filters = new Filters();
Filter filter = new Filter("CUBEOBJECT_ID", "", cubeObjId);
filters.add(filter);
```

```
COBuildStatus coBuildStatus =
rom.getCubeObjectBuildStatus(filters, requestorUserInfo);
COBuildingStatus cbStatus =
(COBuildingStatus)coBuildStatus.getCubeStatus();
buildRequestId = cbStatus.getBuildRequestId();
//if build is not yet completed, then only cancel build process
if(!buildRequestId.isEmpty()){
    requestParams.put(InteraConstants.SysParams.REQUESTID_GUID,
    buildRequestId);
    //method to cancel build process
    rom.cancelCubeObjectBuild(requestParams, requestorUserInfo);
}
```

Database connection Management

Get the list of all the DB connections present in the Intellicus Repository

This Java API is used to get the list of all the DB connections present in the Intellicus Repository.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/DBConnection Mangement/GetDBConnectionList.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a Layout Manager class object for report layout management related operations

```
LayoutManager lm = new LayoutManager();
```

4. Get the list of All Database Connections.

Method : getDBConnectionList

```
public java.util.HashMap getDBConnectionList(UserInfo userInfo)
       throws LayoutHandlerException
```

This method returns the list of report engine to database connection names.

Parameters:

- **UserInfo:** UserInfo For authorization.

Returns:

HashMap of Database Connection names and Driver details as ArrayList.

```
HashMap map=lm.getDBConnectionList(requestorUserInfo);
```

Create DB Connection in the Intellicus Repository

This Java API is used to create a new Database Connection in the Intellicus Repository.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/DBConnection Mangement/CreateDBConnection.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a Layout Manager class object for report layout management related operations

```
LayoutManager lm = new LayoutManager();
```

4. Set url for DBDriver.

```
DBDriver driverDB = new DBDriver();

//Making an object of hashmap for DBDriver
HashMap hMap = new HashMap();

String server = "192.168.33.52";
String provider = "MSSQL";
String port = "1433";
String database = "M90";
String driverVersion = "2005";

driverDB.setUrl("jdbc:sqlserver://192.168.33.52:1433;databaseName
=M90;selectMethod=cursor;user=sa;password=intellicus");
driverDB.setProvider(provider);

hMap.put("SERVER", server);
hMap.put("PORT", port);
hMap.put("DATABASE",dataBase);
hMap.put("DRIVERVERSION",driverVersion);

driverDB.setAttrHash(hMap);
```

5. Create an instance of DB Connection and set related properties

```
String userId = "sa";
String passwrd = "123456";
String connName = "MyConnection";
```

```
//Making an object of DBConnection
DBConnection dbConn = new DBConnection();
dbConn.setDbDriver(driverDB);
dbConn.setUserId(userId);
dbConn.setPassword(passwrd);
dbConn.setConnectionName(connName);
```

6. Add DB Connection to Report Server.

Method : addReportServerConnection

```
public java.lang.String addReportServerConnection
          (DBConnection dbConnection, UserInfo userInfo)
          throws LayoutHandlerException
```

This method adds a new Report Server Connection in the Repository.

Parameters:

- **DbConnection:** DBConnection object which needs to be added.
- **UserInfo:** The UserInfo object.

Returns:
String: The status of operation returned by the Report Engine

```
lm.addReportServerConnection(dbConn, requestorUserInfo);
```

Create DB Connection for File Data Source

This Java API is used to create a new Database Connection in the Intellicus Repository for File Type Data Source.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/DBConnection Mangement/CreateDBConnectionFromFileSource.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a Layout Manager class object for report layout management related operations

```
LayoutManager lm = new LayoutManager();
```

4. Set url for DBDriver.

```
DBDriver driverDB = new DBDriver();
```

```
//Making an object of hashmap for DBDriver
HashMap hMap = new HashMap();

driverDB.setUrl("jdbc:FS:////192.168.33.93/FilesForConnection/csv
");
driverDB.setProvider("FILES");
//set driver type
hMap.put("DRIVERTYPE","NETWORK_PATH");
driverDB.setAttrHash(hMap);
```

5. Create an instance of DB Connection and set related properties

```
String connName = "FileConnection";

//Making an object of DBConnection
DBConnection dbConn = new DBConnection();
dbConn.setDbDriver(driverDB);
dbConn.setConnectionName(connName);
//set username
dbConn.setUserId("UserID");
//set password
dbConn.setPassword("pwd123");
//set Initial Connections
dbConn.setInitialConnections("2");
//Set incremental size
dbConn.setIncrementSize("4");
//Set Maximum number of Connections
dbConn.setMaxConnections("8");
//Set true for Read Only Connection or else set false
dbConn.setIsReadOnly(false);
//Set true to set it as Default Connection
dbConn.setIsDefault(false);
//To anable/disable Metadata Cache for the Connection
dbConn.setMetaDataCacheEnabled(true);
```

6. Add DB Connection to Report Server.

Method : addReportServerConnection

```
public java.lang.String addReportServerConnection
          (DBConnection dbConnection, UserInfo userInfo)
          throws LayoutHandlerException
```

This method adds a new Report Server Connection in the Repository.

Parameters:

- **DbConnection:** DBConnection object which needs to be added.
- **UserInfo:** The UserInfo object.

Returns:
String: The status of operation returned by the Report Engine

```
lm.addReportServerConnection(dbConn, requestorUserInfo);
```

Delete DB Connection from the Intellicus Repository

This Java API is used to delete an existing Database Connection in the Intellicus Repository.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/DBConnection Mangement/DeleteDBConnection.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a Layout Manager class object for report layout management related operations

```
LayoutManager lm = new LayoutManager();
```

4. Get the DB Connection list and iterate through each element of list so as to get the given connection i.e. to be deleted

If the connection exists, then delete it.

Method: deleteReportServerConnection

```
public java.lang.String deleteReportServerConnection
(DBConnection dbConnection, UserInfo userInfo)
       throws LayoutHandlerException
```

This method deletes the given report server connection from the repository.

Parameters:

- **DbConnection:** DBConnection object which needs to be deleted.
- **UserInfo:** The UserInfo object.

Returns:

String: The status of operation returned by the Report Engine.

Part of Sample Code implementing "deleteReportServerConnection":

```
map=lm.getDBConnectionList(requestorUserInfo);
Set s=map.keySet();
Iterator itr=s.iterator();
while(itr.hasNext())
{
  ArrayList l=(ArrayList)(map.get(itr.next()));
  //Iterating through the List
  for(int i=0;i<l.size();i++)
  {
        Object o=l.get(i);

        if("class com.intellica.client.common.DBConnection"
        .equalsIgnoreCase(String.valueOf(o.getClass())))
        {
              DBConnection dbConn=(DBConnection)l.get(i);

              // Check whether the Connection exists or not
              if(connectionName.equalsIgnoreCase
                    (dbConn.getConnectionName()))
              {
              try
              {
              //The method delete given Report Server Connection
              lmanager.deleteReportServerConnection(dbConn,
                                    requestorUserInfo);
              }catch(LayoutHandlerException lhException)
              {
                    lhException.printStackTrace();
              }
              }//end inner if
        }//end Outer if
  }//end for
}//end while
```

Audit Log

Get Audit Detail

This Java API is used to get the Audit detail information from the Intellicus Repository.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/Audit Log/GetAuditDetail.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create an Audit Manager controller class object for audit management related operations.

```
AuditManager am= new AuditManager();
```

4. Set the filter for values fields.

```
// prepare parameters for filtering
Date fromDate = new Date(2006-1900 ,2-1,01);
Date toDate = new Date(2006-1900,3-1,9);
String reportName  = "Product";
String username = "Admin";
String reportId = "91FEE269-3AEE-C23D-6F04-7A9979BEBE09";
String categoryId  = "Test";

Filter filter=new Filter();
filter.setFilterField("REPORTNAME",reportName);
filter.setFilterField("CATEGORYID",categoryId);
filter.setFilterField("FROMDATE",fromDate);
filter.setFilterField("TODATE",toDate);
filter.setFilterField("REPORTID",reportId );
filter.setFilterField("USERID",userName);
```

5. Get the Audit Log details based on filter applied.

Method : getAuditLogList

```
public java.util.ArrayList getAuditLogList
       (java.lang.String fromDate,
        java.lang.String toDate,
        java.lang.String reportName,
        java.lang.String userName,
        java.lang.String reportId,
        java.lang.String categoryId,
        UserInfo userInfo)
 throws AuditManagerException
```

This method will provide the list containing audit log information.

Parameters :

Filter: Filter which can take request filters from Enums.Filters.AuditLog
i.e.
Enums.Filters.AuditLog.REPORTNAME
Enums.Filters.AuditLog.CATEGORYID
Enums.Filters.AuditLog.FROMDATE

Enums.Filters.AuditLog.TODATE
Enums.Filters.AuditLog.REPORTID
Enums.Filters.AuditLog.USERID
userInfo: The User Information Object

Returns:
ArrayList of Audit Log information of various reports.Each element of list is object of class AuditLogData.

```
ArrayList arrayList=am.getAuditLogList(filter,requestorUserInfo);
```

Delete Audit Detail

This Java API is used to delete Audit Log information.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/Audit Log/DeleteAuditLog.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a Audit Manager controller class object for audit management related operations.

```
AuditManager am= new AuditManager();
```

4. Set the filter for values fields that are to be deleted in Audit Details.

```
Date fromDate = new Date(2006-1900 ,2-1,01);
Date toDate  = new Date(2006-1900,3-1,9
String reportName  = "Product";
String username = "Admin";
String categoryId  = "Test";
String reportId    = "91FEE269-3AEE-C23D-6F04-7A9979BEBE09";

Filter filter=new Filter();
filter.setFilterField("REPORTNAME",reportName);
filter.setFilterField("CATEGORYID",categoryId);
filter.setFilterField("FROMDATE",fromDate);
filter.setFilterField("TODATE",toDate);
filter.setFilterField("REPORTID",reportId );
filter.setFilterField("USERID",userName);
```

5. Delete the Audit Log List as per filter

Method : deleteAuditLogList

```
public void deleteAuditLogList(Filter filter, UserInfo userInfo)
       throws AuditManagerException
```

This method will delete audit log information from the Report Engine.

Parameters:

- **Filter:** Filter which can take request filters from Enums.Filters.AuditLog

i.e.
Enums.Filters.AuditLog.REPORTNAME
Enums.Filters.AuditLog.CATEGORYID
Enums.Filters.AuditLog.FROMDATE
Enums.Filters.AuditLog.TODATE
Enums.Filters.AuditLog.REPORTID
Enums.Filters.AuditLog.USERID
UserInfo: The User Information Object

```
am.deleteAuditLogList(filter,requestorUserInfo);
```

Data Masking

Save Dtaa Masking

This Java API is used to save the data masking details.
Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/Data Masking/SaveData Masking.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create an instance of SecurityManager.

```
SecurityManager sMgr = SecurityManager.getInstance();
```

4. Set the various required details, like Connection name, table name, column name, masking character etc.

```
MaskedData maskedData = new MaskedData();
ArrayList<ConnMaskedDetails>        connList        =        new
ArrayList<ConnMaskedDetails>();
ConnMaskedDetails connMaskedDetailsObj = new ConnMaskedDetails();
connMaskedDetailsObj.setConnectionName("DemoReportDB");
//Connection Name
ArrayList<DBSchema> dbSchemaList = new ArrayList<DBSchema>();
DBSchema dbSchemaObj = new DBSchema();
DBMaskedEntities dbMaskedEntitiesObj = new DBMaskedEntities();
DBMaskedEntity dbMaskedEntityObj = new DBMaskedEntity();
dbMaskedEntityObj.setDBEntityName("BRANCH"); //Table name
dbMaskedEntityObj.setDBEntityType("0");
```

```
Column columnObj = new Column();
columnObj.setColumnId("365372309");  //Some  GUID  for  setting
Column Id
columnObj.setColumnName("BRANCH_ID"); //column Name
columnObj.setColumnMaskChar("#"); //Mask character
columnObj.setMaskLevel("1");  //For  Masking  only  on  provided
connection, 0 for Mask for All Connections
columnObj.setMaskType(0); //for masking completely set o, and for
masking partially set 1
columnObj.setColumnOpcode("ADD");//ADD for adding data masking on
a  column,  UPDATE  for  updating  data  masking  on  already  masked
column

ArrayList<ColGrant> colGrantList = new ArrayList<ColGrant>();
ColGrant colGrantObj = new ColGrant();
colGrantObj.setOrgId("HostOrg");  //Organization  for  exceptional
users and roles
colGrantObj.addRoleId("Admin");    //Role
colGrantObj.addUserId("John");     //User
colGrantObj.addUserId("Mary");     //User
colGrantList.add(colGrantObj);
columnObj.setColGrantList(colGrantList);

dbMaskedEntityObj.addColumn(columnObj);
dbMaskedEntitiesObj.addMaskedEntity(dbMaskedEntityObj);
dbSchemaObj.setDbMaskedEntities(dbMaskedEntitiesObj);
dbSchemaList.add(dbSchemaObj);

connMaskedDetailsObj.setDbSchemaList(dbSchemaList);
connList.add(connMaskedDetailsObj);
maskedData.setConnList(connList);
```

5. Save the data masking details.

Method : saveColsSecurityInfo

```
public void saveColsSecurityInfo(String strCLRSaveList, UserInfo
userInfo)
       throws ISecurityException
```

Parameters :

maskedData : Object of {@link com.intellica.client.security.MaskedData} with all the column masking* details filled.
userInfo: userInfo object keeping the detail about the current user

Get Masked Columns

This Java API is used to get the Masked columns details for specified connection

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/Audit Log/GetMaskedColumns.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.

3. Create an instance of SecurityManager.

```
SecurityManager sMgr = SecurityManager.getInstance();
```

4. Get the masking details.

Method: getColsSecurityInfo

Returns the security information of all columns as ArrayList of ColSecurityInfo class.

```
public MaskedData getColsSecurityInfo(Filter filter, UserInfo
userInfo)
        throws ISecurityException
```

Parameters :

filter : Filter object to get the masked details of given filter creteria only.

- Possible values of filter are {@link com.intellica.client.common.Enums.Filters.ColumnMaskDetails}.

userInfo: userInfo object keeping the detail about the current user

5. Below is the code snippet for iterating through the masked column list-

```
MaskedData maskedData = sMgr.getColsSecurityInfo(filter,
requestorUserInfo);

ArrayList connList = maskedData.getConnList();
//Iterating over the connections to get masked columns for that
connection
for(int i=0;i<connList.size();i++){
  ConnMaskedDetails connMaskedDetailObj =
(ConnMaskedDetails)connList.get(i);
  ArrayList dbSchemaList = connMaskedDetailObj.getDbSchemaList();
  for(int j=0;j<dbSchemaList.size();j++){
        DBSchema dbSchemaObj = (DBSchema)dbSchemaList.get(j);
        DBMaskedEntities dbMaskedEntitiesObj =
dbSchemaObj.getDbMaskedEntities();
```

```
        ArrayList dbMaskedEntityList =
dbMaskedEntitiesObj.getMaskedEntityList();
        for(int k=0;k<dbMaskedEntityList.size();k++){
              DBMaskedEntity dbMaskedEntityObj =
(DBMaskedEntity)dbMaskedEntityList.get(k);
              System.out.println("Table Name =
"+dbMaskedEntityObj.getDBEntityName());  //Table name
              System.out.println("----------------------------");
              ArrayList columnList =
dbMaskedEntityObj.getColumnList();
              //Iterating over the masked column list
              for(int
l=0;l<dbMaskedEntityObj.getColumnListSize();l++){
                    Column colObj =
(Column)dbMaskedEntityObj.getColumn(l);
                    System.out.println("Column Name =
"+colObj.getColumnName()+", MaskType = "+colObj.getMaskType()+",
MaskLevel = "+colObj.getMaskLevel()+", MaskChar =
"+colObj.getColumnMaskChar());
              }
  System.out.println("========================================");
        }
  }
}
```

ReporServerProperty

GetReportEnginePortAndIP

This Java API is used to get Report Engine's IP Address and Port number of the Report Engine.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/ ReportServerProperty/ GetReportEnginePortAndIP.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a Layout Manager class object for report layout management related operations

```
LayoutManager lm = new LayoutManager();
```

4. Get the IP of Report Engine.

Method : getReportEngineIP

```
public java.lang.String getReportEngineIP()
```

Get report Engine IP, sets through any of the constructor or if not then returns the default value from ConfigManager class.

Returns:
String: Report Engine IP

```
// method used to get the IP Address of Report Engine
// Returns ReportEngine IP as a String value.
String reportEngineIP=lm.getReportEngineIP();
```

5. Get the port of Report Engine.

Method : getReportEnginePort

```
public int getReportEnginePort()
```

Get report Engine Port, sets through any of the constructor or if not then returns the default value from ConfigManager class.

Returns:
String : report Engine Port

```
  // method used to get the Port of Report Engine
  // Returns report Engine Port as Integer value
  int reportEnginePort=lm.getReportEnginePort();
```

GetReportServerProperty

This Java API is used to get SMTP Server Values of the Report Engine.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/ ReportServerProperty/GetReportServerProperty.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a SecurityManager class object for getting the controller information for all Administration related operations

```
SecurityManager sMgr=SecurityManager.getInstance();
```

4. Get the Property values of Report Server.

Method : getReportServerPropValue

```
public java.lang.String getReportServerPropValue
```

```
          (java.util.ArrayList reportServerPropList,
           java.lang.String key,UserInfo userInfo)
    throws ClientException
```

This method gets the value of Report Server Property from the Report Server for given value of key

Parameters:

- **UserInfo:** The UserInfo object.
- **ReportServerPropList:** ArrayList of all Report Server Properties
- **key:** Name of property whose value is required

Returns:
String representing required property value. If no property is identified with given key then this Method returns null.

Part of Sample Code implementing "getReportServerPropValue" :

```
//This returns arraylist of values of Report Server properties
ArrayList arrList =
sMgr.getReportServerPropDetails(requestorUserInfo);
```

5. Get all the property values of Report Server.

```
//Set the key for the server property.This key is the name     of
the property as specified in the ReportEngine.properties       at
< Intellicus_Install_path >\ReportEngine\Config folder
String key="SMTP SERVER";//Value of Report Server property

String value=sMgr.getReportServerPropValue(arrList ,key,
                                    requestorUserInfo);
System.out.println ("SMTP_SERVER Value : "+value);

key="LISTENER_PORT";
value=sMgr.getReportServerPropValue(arrList ,key,
                         requestorUserInfo);
System.out.println (" LISTENER_PORT Value : "+value);

key="DATABASE_CONNECTION_TIMEOUT";
value=sMgr.getReportServerPropValue(arrList ,key,
                               requestorUserInfo);
System.out.println ("DATABASE_CONNECTION_TIMEOUT Value :
  "+value);
```

```
key="SECURITY_FEATURES";
value=sMgr.getReportServerPropValue(arrList ,key,
                                requestorUserInfo);
System.out.println ("SECURITY_FEATURES Value : "+value);

key ="AUDIT_LOG";
value=sMgr.getReportServerPropValue(arrList ,key,
                                    requestorUserInfo);
System.out.println ("AUDIT_LOG Value : "+value);

key = "QUEUE_SIZE";
value=sMgr.getReportServerPropValue(arrList ,key,
                                    requestorUserInfo);
System.out.println ("QUEUE_SIZE Value : "+value);

key = "REMOTE_SESSION_TIMEOUT";
value=sMgr.getReportServerPropValue(arrList ,key,
                                requestorUserInfo);
System.out.println ("REMOTE_SESSION_TIMEOUT Value : "+value);

key = "RTF_FIELD_CONTROL_MAP";
value=sMgr.getReportServerPropValue(arrList ,key,
                                    requestorUserInfo);
System.out.println ("RTF_FIELD_CONTROL_MAP Value : "+value);

key = "DATA_SOURCE_FETCH_SIZE";
value=sMgr.getReportServerPropValue(arrList ,key,
                  requestorUserInfo);
System.out.println ("DATA_SOURCE_FETCH_SIZE Value : "+value);

key = "AUDITLOG_PURGE_TIME";
value=sMgr.getReportServerPropValue(arrList ,key,
                                    requestorUserInfo);
System.out.println ("AUDITLOG_PURGE_TIME Value : "+value);

key = "CACHE_PURGE_TIME";
value=sMgr.getReportServerPropValue(arrList ,key,
                                requestorUserInfo);
System.out.println ("CACHE_PURGE_TIME Value : "+value);

key = "AUTHORIZATION_CACHE_TIMEOUT";
value=sMgr.getReportServerPropValue(arrList ,key,
                                requestorUserInfo);
System.out.println("AUTHORIZATION_CACHE_TIMEOUT Value :
                                                "+value);
```

```
key = "SCHD_JOB_DISPATCH_QUEUE_SIZE";
value=sMgr.getReportServerPropValue(arrList ,key,
                                    requestorUserInfo);
System.out.println("SCHD_JOB_DISPATCH_QUEUE_SIZE Value :
                                    "+value);
```

SetReportServerProperty

This Java API is used to set the SMTP Server Values of Report Engine.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/ ReportServerProperty/ SetReportServerProperty.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a SecurityManager class object for getting the controller information for all Administration related operations

```
SecurityManager sMgr=SecurityManager.getInstance();
```

4. Set the Property values of Report Server by putting them in a Hashmap.

```
HashMap ServerPropHMap= new HashMap();

  String key="SMTP_SERVER";
  String value="192.168.100.20";
  ServerPropHMap.put(key,value);

  key="DATABASE_CONNECTION_TIMEOUT";
  value="800";
  ServerPropHMap.put(key,value);

  key="SECURITY_FEATURES";
  value="enabled";
  ServerPropHMap.put(key,value);

  key="QUEUE_SIZE";
  value="900";
  ServerPropHMap.put(key,value);

  key="LOG";
  value="../logs";
  ServerPropHMap.put(key,value);
```

```
key="PAGE_CHUNKSIZE";
value="10";
ServerPropHMap.put(key,value);

key="LOG_LEVEL";
value="INFO";
ServerPropHMap.put(key,value);

key="USER_THREADS";
value="5";
ServerPropHMap.put(key,value);

key="REPOSITORY_CACHE_TIMEOUT";
value="15";
ServerPropHMap.put(key,value);

key="EXIT_ON_ERROR";
value="disable";
ServerPropHMap.put(key,value);
```

5. Save these property values set in last step for Report Server

Method : saveReportServerProperties

```
public java.lang.String saveReportServerProperties
      (java.util.HashMap reportServerPropMap,
       UserInfo userInfo)
throws ISecurityException
```

This method saves the modified server properties and returns the status sent by the Report Server.

Parameters:

- **ReportServerPropMap:** HashMap which keeps the property name as key and its its concerning value as the value.
- **UserInfo:** The UserInfo object.

Returns:

String: Having the status return by the Report Engine after modifying the property file.

Part of Sample Code implementing "setReportServerPropValue" :

```
sMgr.saveReportServerProperties(ServerPropHMap,
                                requestorUserInfo);
```

ReporServerConnectivity

TestServerConnectivity

This Java API is used to check whether the Report Server is running on the specified IP and Port or not.

Refer to Intellicus_Install_Path>/SampleCodes/Java APIs/TestServerConnectivity/ TestServerConnectivity.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Create a Security controller class object using its factory for user management related operations

```
SecurityManager sMgr=SecurityManager.getInstance();
```

3. Get the Security Mode of Report Engine.

Method : getSecurityMode

```
public boolean getSecurityMode()
       throws ISecurityException
```

This method returns the security mode read from the report engine.

Returns:
If the security mode is on on the report engine, it returns true. In other cases, it returns false.

```
boolean securityMode    = false;

//If it gives Exception, then failed to connect Report Server
securityMode=sMgr.getSecurityMode();
```

User Preferences

GetUserPreferences

This Java API is used to get the default User Preferences set by the User.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/User Preferences/GetUserPreferences.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a PersonalizationManager class object for UserPrefernces and Dashboard related settings.

```
PersonalizationManager pm=new PersonalizationManager();
```

4. Get the User Preferences in the instance of Preferences.

Method: getUserPreferences

```
public Preferences getUserPreferences
       (UserInfo userInfo)
throws PersonalizationHandlerException
```

This method returns Preferences for the given user.

Parameters:

- **UserInfo:** The Information about the user to be used for authorization. Also the user whose Preferences will be returned.

Returns:
Preferences for the user.

```
Preferences userPref = null;
userPref=  pm.getUserPreferences(requestorUserInfo);
```

5. Get all the User Preferences

```
//Returns the defaultConnection
String defaultCon = userPref.getDefaultConnection();

//Returns the user's preferred default portal theme
String  portalTheme = userPref.getDefaultPortalTheme();

//Method returns default stylesheet of the user
String defaultStylesheet = userPref.getDefaultStylesheet();

//Method returns the user's preferred deliver location
String deliveryLocation = userPref.getDeliveryLocation();

//Method returns the user's preferred report format
String defaultFormat = userPref.getFormat();
```

```
 //Method returns the user's preferred language
 String prefLanguage = userPref.getPrefLanguage();

//Method returns the number of recent reports to be shown
int reportCount = userPref.getReportCount();

//Method returns the Boolean value whether to show the inbox
Boolean showInbox = userPref.getShowInbox();

//This Method returns default template name used for all
   reports
String templateName = userPref.getTemplateName();

// Method returns the Boolean value whether to show the recent
   reports or not
Boolean showRecentReports = userPref.getShowRecentReports();

System.out.println("Default Connection  :  "+defaultCon);
System.out.println("Default Portal Theme  :  "+portalTheme);
System.out.println("Default Stylesheet:"+defaultStylesheet);
System.out.println("Delivery Location  :  "+deliveryLocation);
System.out.println("Default Format  :  "+defaultFormat);
System.out.println("Default Preferred Language  :
   "+prefLanguage);
System.out.println("Report Count  :  "+reportCount);
System.out.println("Show Inbox  :  "+showInbox);
System.out.println("Template Name  :  "+templateName);
System.out.println("Show Recent Reports  :
             "+showRecentReports);
```

setUserPreferences

This Java API is used to set the default User Preferences by the User.

Refer to <Intellicus_Install_Path>/SampleCodes/Java APIs/User Preferences/SetUserPreferences.java for sample code of this use case.

Steps:

1. Initialize Report Client.
2. Initialize Requestor UserInfo.
3. Create a PersonalizationManager class object for UserPrefernces and Dashboard related settings.

```
PersonalizationManager pm=new PersonalizationManager();
```

4. Get the User Preferences in the instance of Preferences.

```
Preferences pref = new Preferences();
pref=  pmanager.getUserPreferences(requestorUserInfo);
```

5. Set the new value for User Preferences.

```
int reportCount = 12;
String templateName = "Beach";
String email = "hostUser@hostOrg.com";
String defConnection = "MyConnection";

pref.setReportCount(reportCount);
pref.setTemplateName(templateName);
pref.setShowRecentReports(true);
pref.setDefaultConnection(defConnection);
pref.setDefaultPortalTheme("Default");
pref.setEmail(email);
pref.setFormat(InteraConstants.ReportFormats.PDF);
pref.setPrefLanguage("EN_US");
```

6. Update these User Preferences

Method: updateUserPreferences

```
public Preferences updateUserPreferences
       (Preferences userPref,
        UserInfo userInfo)
throws PersonalizationHandlerException
```

This method updates the user preferences to the repository.

Parameters:

- **UserPref:** The user preferences prefernces details
- **UserInfo:** userInfo object for authorization.

Returns:
Preferences

```
//Method used to update the user preferences to the repository
pmanager.updateUserPreferences(pref, requestorUserInfo);
```

Callback API

Call Back mechanism has three implementations.

- Global Security Filtering
- Authentication Check
- Callback Events

Intellicus allows its customers to enable the security filtering of their report data. The filtering is typically on the basis of any user credential attributes like User ID, Org ID etc. This allows Intellicus customers to show only filtered records to a given user from their database.

The filtering is configured at organization level which enables the filtering on the data in the reports run by the users of that organization.

If a user who belongs to that organization runs a report, Intellicus would either apply filtering it self or call customer's code depending on the configuration made.

This should be noted that during the report execution filtering is applied on all the relevant SQLs run, which include Report SQLs, Crosstab or Chart SQLs, Parameter SQLs etc.

Intellicus allows its customers to apply user authentication check.

Intellicus provides support for calling back the client's plug-in code impelemented for various types of callback events for Report, Connection, UMM etc operations.

SQL Filter

Configuration

Security Filtering can be done in following two ways:

1. **Intellicus**: Using Intellicus, SQL filtering requires administrator to provide the filter column name which must be present in the database tables entities as an attribute and depending on which Intellicus would add a where clause in all the data fetching SQLs. (In the sample this column is WORKSPACEID). This way, the SQl is replaced and records are filtered based on given attribute.

2. **Callback**: Intellicus supports callback SQL Filtering where Intellicus calls custom code to apply the filtering on all the data fetching SQLs.

Sample Schema

Let's assume there are 50 tables, in which 45 tables have a field to separate row data.

Tables: COMPANY, CUSTOMER, CONTACT.

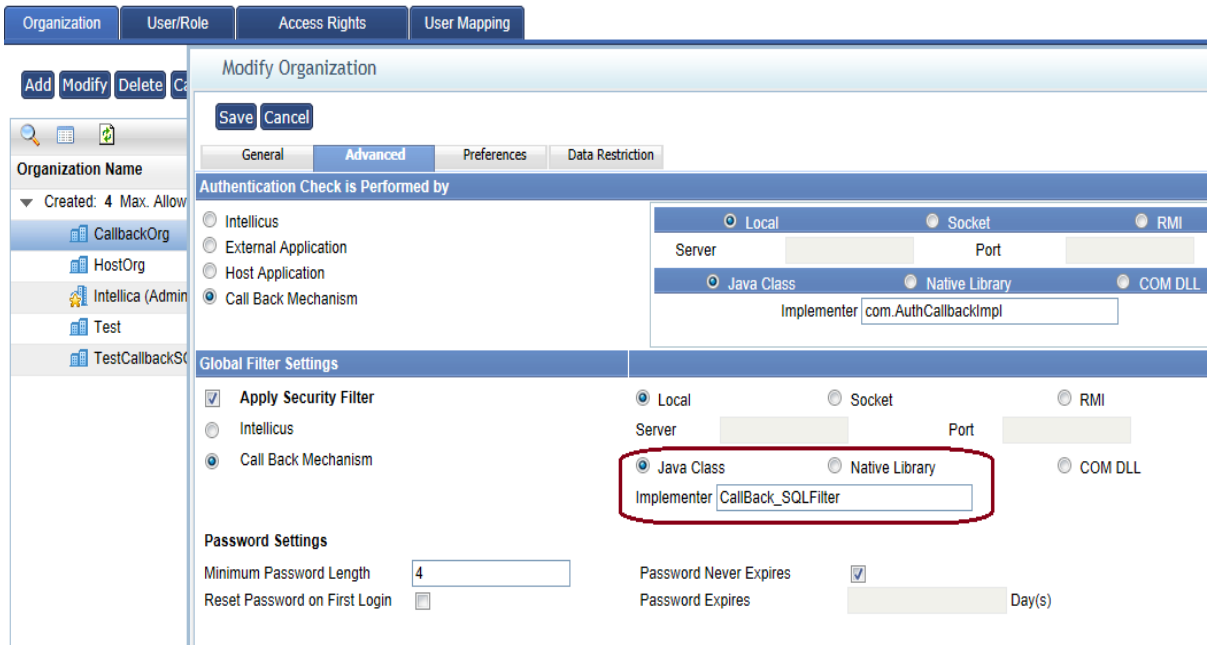
All these tables have a column WORKSPACEID which is used to divide rows amongst users.

Common Tables: CITY, COUNTRY

These tables store generic data and doesn't need filtering.

Configuring SQL Filtering Callback Options

Customer system administrator configures the callback SQL Filtering. For configuration the administrator will go to organization screen where he/she enables callback SQL filtering.



Customer administrator enables security filtering and chooses Call Back Mechanism as the option between Intellicus and Call Back Mechanism.

Using call back security filtering requires administrator to provide the following:

4. Filtering mode among Local, Socket or RMI
5. Implementer class name which is a Java, C, or COM Class.

Callback SQL Filtering

Code implements Intellicus Call back Filtering interface

If Intellicus SQL filtering is not what customer wants, Intellicus also supports call back SQL Filtering where Intellicus calls customer's code to apply the filtering on all the data fetching SQLs.

For Intellicus to call customers code customer will have to provide the implementation of the call back interface exposed by Intellicus.
Intellicus supports call back using various technologies:

Local Java

- **Implementer:** Java Class
- **Call Type:** In Memory

You provide Java implementation of Intellicus sql filtering call back interface and Intellicus calls it at runtime. The custom code library must be placed in Intellicus Report Server class path before starting the server.

Local COM

- **Implementer:** COM Dll written in VB, VC++
- **Call Type:** JNI COM call to Dll.

You provide a COM DLL implementation of the Intellicus SQL filtering call back COM interface. The report Server should be running on a Windows Platform.

Remote Call back

- **Implementer:** COM Dll written in VB, VC++
- **Call Type:** TCP Call to DLL Invoker and JNI Call

You provide a COM DLL implementation of the Intellicus SQL Filtering Call back interface. The report Server is running on non Windows Platform, thus can not support COM.

For the implementation of the interface Customer gets the interface from Intellicus.

Interface Details

For Java call back activities, Intellicus provides the interface in the form of a jar file, iCallback.jar.

The Jar contains the interface exposed by Intellicus, ISQLFilter.

Customer would implement the methods of the interface and provide the implementation class, normally, in the form of a jar file. The jar would be deployed in the Intellicus Report Server class path for Report Server to make the calls.

Equally, Intellicus provides IntellicusContext.dll for COM DLL implementation. The implementer COM DLL should be appropriately registered with the windows registry.

1. Implement the interface ISQLFilter

```
public class CallBack_tables implements ISQLFilter
```

2. Initialize the class instance using initGlobal().

```
public void initGlobal() throws Exception{
  System.out.println("---inside initGlobal()--------");
}
```

Call Sequence

Below are the methods of the call back filtering interface and the sequence in which they should be called:

Please refer to related Java doc for exact signature of the interface methods.

1) initGlobal()

Set the credentials and other information (helps custom class to initiate config files and database connections etc.).

2) int getFilterType()

Intellicus gets the filter type you want to implement:

Return Value = **1** or **FILTER_TYPE_SQL**
Type: **SQL**
Complete SQL statement will be passed for manipulation by your code and collected back.
Method for filter call: **getFilteredSQL()**

Return Value = 2 or **FILTER_TYPE_TABLES**
Type: **TABLES**
List of TABLES will be passed for manipulation by your code and collected back.
Method for filter call: **getTablesFilters()**

3) setOrgID(String orgID)

A series of setters are called for setting the context and credentials of the reporting user. Reporting user's Intellicus OrgiD.

4) setUserID(String userID)

Reporting user's Intellicus UserID.

5) setPassword(String passwd)

Reporting user's Intellicus Password.

6) setSessionID(String sessionID)

Reporting user's Intellicus sessionID.

7) setSD(String sd)

Reporting user's SecurityDescriptor. (Custom tag)

8) setCustomerID(String customerID)

Reporting user's Customer ID. (Custom tag)

9) setLocation(String location)

Reporting user's Location ID. (Custom tag)

10) setLocale(String locale)

Reporting user's Locale. (Custom tag)

11) setTimestamp(String timestamp)

Reporting request Time stamp (Custom tag)

12) setDBName(String dbName)

Get the filtered sql if filter type is SQL Filter Call

13) String getFilteredSQL(String sql)

Intellicus provides the original SQL to your code and gets the modified SQL back.

OR

13) String[] getTablesFilters(String[] Entities)

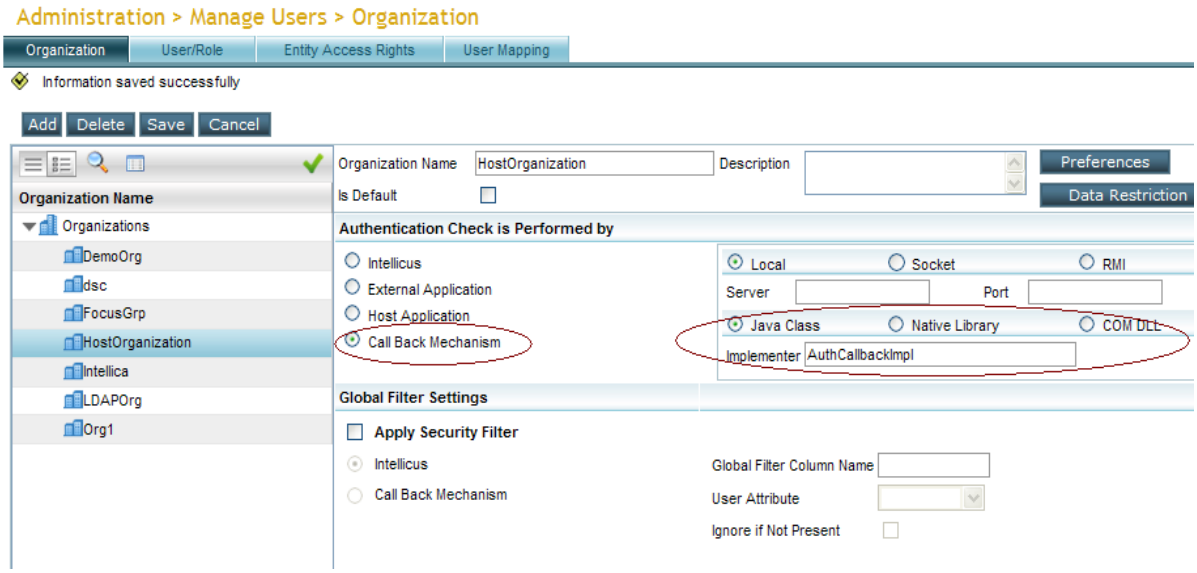
Intellicus provides the Entity names used in the SQL to your code and gets back the filters list to be replaced at the place of those Entities or Implicit Views to replace the table names.

Note: Please refer sample code available at <Intellicus_Install_Path>\SampleCodes\CallBack APIs\CallBack Filtering

Authentication Check

Configuring Authentication Check

System Administrator configures Intellicus Call Back Authentication check. Customer system administrator configures the Intellicus call back authentication mechanism. For configuration the administrator will go to organization screen where he/she selects the option for call back authentication check.



Customer administrator selects the option of Call Back Mechanism for Authentication check.

Using call back authentication check requires administrator to provide the following:

1. Filtering mode among Local, Socket or RMI
2. Implementer class name which is a Java, C, or COM Class.

Callback Authentication Check

Your Code implements Intellicus Call back authentication interface.

If Intellicus authentication checks is not what customer wants, Intellicus also supports call back authentication check where Intellicus calls customer's code to apply the authentication check.

For Intellicus to call customers code customer will have to provide the implementation of the call back interface exposed by Intellicus.

Intellicus supports call back using various technologies:

Local Java

- **Implementer:** Java Class
- **Call Type:** In Memory

You provide Java implementation of Intellicus authentication call back interface and Intellicus calls it at runtime. The custom code library must be placed in Intellicus Report Server class path before starting the server.

Local COM

- **Implementer:** COM Dll written in VB, VC++
- **Call Type:** JNI COM call to Dll.

You provide a COM DLL implementation of the Intellicus authentication call back COM interface. The report Server should be running on a Windows Platform.

Remote Call back

- **Implementer:** COM Dll written in VB, VC++
- **Call Type:** TCP Call to DLL Invoker and JNI Call

You provide a COM DLL implementation of the Intellicus authentication Call back interface. The report Server is running on non Windows Platform, thus can not support COM.

For the implementation of the interface Customer gets the interface from Intellicus.

Interface Details

For Java call back activities Intellicus provides the interface in the form of a jar file, iCallback.jar.

The Jar contains the interface exposed by Intellicus, IAuthenticate.

Customer would implement the methods of the interface and provide the implementation class, normally, in the form of a jar file. The jar would be deployed in the Intellicus Report Server class path for Report Server to make the calls.

Equally, Intellicus provides IntellicusContext.dll for COM DLL implementation. The implementer COM DLL should be appropriately registered with the windows registry.

Sample Implementation Code

Steps:

1. Impletment the interface IAuthenticate.

```
public class AuthCallbackImpl implements IAuthenticate
```

2. Initialize the class instance using initGlobal().

```
public void initGlobal() throws Exception

public boolean authenticate() throws Exception
{
  System.out.println("calling authenticate() method");

  //This is the JDBC url to connect to host application database.
  String url = "jdbc:oracle:thin:@192.168.100.22:1521:test";

  try{
        //configuring the database.
        Connection = DriverManager.getConnection( url, "scott",
        "tiger");
        if(connection==null){
              System.out.println("failed to connect to database");
        }
        else{
              ResultSet rs = null;
              String passwordFromDB=null;
              Statement stmt = connection.createStatement();
              //The EMP_ID is set through the portal the userId
              that is to be taken from the database.
              rs = stmt.executeQuery("select EMP_ID,PASSWD from
                    EMP where EMP_ID ='"+this.userID+"'");
              //The result set has the field PASSWD to be used as
              //password,that is matched after then only user will
              //be authenticated.
              while(rs.next()){
                    //The password is also matched with the data
                    //present in the database.The password
                    //is also provided through the portal.
                    passwordFromDB=rs.getString("PASSWD");

        System.out.println("Password:*"+passwdEnteredByUser+"*");
        System.out.println("password from
                              database:*"+passwordFromDB+"*");
```

```
        if(passwordFromDB.equals(passwdEnteredByUser))
        {
           System.out.println("user AUTHENTICATED");
           return true;
        }
  }
}
}catch( Exception e )
{
  System.out.println ("message:"+e.getMessage());
e.printStackTrace();
}
return false;
}
```

Call Sequence

Below are the methods of the call back filtering interface and the sequence in which they should be called.

Please refer to related Java doc for exact signature of the interface methods.

1) initGlobal()

Set the credentials and other information (helps custom class to initiate config files and database connections etc.).

2) authenticate()

Authentication implementation will be in this method.

3) setOrgID(String orgID)

A series of setters are called for setting the context and credentials of the reporting user. Reporting user's Intellicus OrgiD.

4) setUserID(String userID)

Reporting user's Intellicus UserID.

5) setPassword(String passwd)

Reporting user's Intellicus Password.

6) setSessionID(String sessionID)

Reporting user's Intellicus sessionID.

7) setSD(String sd)

Reporting user's SecurityDescriptor. (Custom tag)

8) setCustomerID(String customerID)

Reporting user's Customer ID. (Custom tag)

9) setLocation(String location)

Reporting user's Location ID. (Custom tag)

10) setLocale(String locale)

Reporting user's Locale. (Custom tag)

11) setTimestamp(String timestamp)

Reporting request Time stamp (Custom tag)

12) setDBName(String dbName)

Get the filtered sql if filter type is SQL.

Note: Please refer sample code available at <Intellicus_Install_Path>\SampleCodes\CallBack APIs\CallBack Authentication

Callback Events

Intellicus supports 'Callback Events'. This feature allows Intellicus to raise particular event on execution of a specific task. The events raised are received by call back code. By receiving all these events users will be able to perform all necessary actions according to their needs.

Java programmers can leverage their existing code by calling them from various Callback events thrown by Intellicus.

- For programming, Intellicus takes the compiled java code.
- Intellicus publishes Interface classes in iCallback.jar
- Implement the interface corresponding to the requested event type.

After compilation of its class and creating its Jar File, place it in the lib folder of ReportEngine i.e. <Intellicus Installed Path>/ReportEngine/lib

Make entries of these class names in eventshandlers.xml placed in Config folder of ReportEngine. Intellicus provides following types of Callback events and their methods:

ReportEvents

1. afterReportExecution
2. beforeReportExecution
3. beforeURLAssign
4. beforeParamsInitialization

Connection Events

1. beforeConnectionGet
2. afterConnectionGet
3. beforeConnectionSubmit

UMM Events

1. afterOrganizationCreate
2. afterRoleCreate
3. afterUserCreate
4. afterUserModify

ReportMgmtEvents

5. afterCategoryAdded
6. afterCategoryModify
7. afterCategoryDelete
8. afterReportAdded

9. afterReportModify
10. afterReportDelete

ROMgmtEvents

1. afterReportObjectAdded
2. afterReportObjectModify
3. afterReportObjectDelete

General configuration

To configure an event,

1. You need to set value of EVENTSHANDLER TYPE tag. This tag is found in configuration file <Intellicus install path>\ReportEngine\Config\ eventshandlers.xml.
2. The name of the IMPLEMENTOR class should be mentioned in the EVENTS configuration XML file.

For example, in case of Report Events,

```
<EVENTSHANDLERS>
  <EVENTSHANDLER TYPE=”REPORTEVENTS”>
        <CALLBACK CALLTYPE="1">
              <IMPLEMENTER TYPE="1">
                    <ATTRS TYPE="1">
                          <ATTR NAME="PATH”>
                                <VALUE>com.mypackage.myclass</VALUE>
                          </ATTR>
                    </ATTRS>
              </IMPLEMENTER>
        </CALLBACK>
  </ EVENTSHANDLER>
</ EVENTSHANDLERS>
```

For each type of callback event there would be one class and corresponding entry in the configuration XML**. eventshandlers.xml** may contain multiple event handlers. Event Handler, if defined in configuration file automatically performs initialization with default implementation.

Report Events

Intellicus provides an interface class "ReportEvents". This interface class enforces the below given methods for various events. Intellicus integrator may write a java class to overwrite the default impelementaion of one or more methods in this interface.

Configuration

For implementing **Report Events**, **EVENTSHANDLER TYPE** should be: REPORTEVENTS.

Following code shows the XML entry having configuration of Report Events.

```
<EVENTSHANDLER TYPE=”REPORTEVENTS”>
  <CALLBACK CALLTYPE="1">
        <IMPLEMENTER TYPE="1">
              <ATTRS TYPE="1">
                    <ATTR NAME="PATH”>
                          <VALUE>com.mypackage.myclass</VALUE>
                    </ATTR>
              </ATTRS>
        </IMPLEMENTER>
  </CALLBACK>
</ EVENTSHANDLER>
```

During callback events process Intellicus Report Server will call these methods:

Object beforeReportExecution(): Gather the callback event information before the execution of a Report from the java class using HashMap as object.

void afterReportExecution(Object eventInfo): Provide the callback event Infomation after the execution of a Report to the java class using HashMap as object.

Object beforeURLAssign(Object eventInfo): Provide the callback event Infomation before the execution of a URL to the java class using HashMap as object and will gather the modified URL from the java class using HashMap as object.

void beforeParamsInitialization(Object eventInfo): Provide the callback event Infomation before the initialization of Parameters while running Report. It basically provides a mechanism of doing some pre-processing on the parameters.

Sample Implementation code

```
import com.impetus.interaj.callback.ReportEvents;

public class SampleReportEventHandlerWithImpl extends
ReportEvents{

  /* Default Constructor.  */

  public SampleReportEventHandlerWithImpl () throws Exception
  {
  }
```

BEFORE REPORT EXECUTION

To get the details from user before report gets executed. This event will be triggered just before the execution of Report in Intellicus.

The following methods of the class will be called in this event.

Object beforeReportExecution()

This method returns an Object which is generally a HashMap object that stores as a key, String that specify the property and as a value, a String that specifies the customizations on that property. The HashMap can have the following fields:

CONNECTION_NAME => Name of the connection.

If a connection object is needed, then provide the name of the connection as a string.

The method should return a HashMap of information as Java class object.
The HashMap is scanned for these key-values.

Event Info Hash Map

Key	Value
CONNECTION_NAME	If you need a connection object other than the connection used to execute the report, then provide the name of the connection as a string. If this key doesn't exist or it is blank, then the connection used for executing the report will be provided in setEventInfo call.
OPERATION_TYPE	The operation performed on the report. **SAVE** = Report was requested for saving by UI or by scheduler. In this case, SAVED_OUTPUT_ID is also available in this HashMap for future use. **EXEC =** The report was requested for viewing on the UI. In this case, SAVE_OUTPUT_ID will be blank or null. **PRINT =** The report was requested for printing on the UI or on the server side. In this case, SAVE_OUTPUT_ID will be blank or null.

USER_INFO	**Hash Map** of USER_INFO.
	
	`HashMap userInfo = (HashMap) eventInfo.get("USER_INFO") ;`
	`String appID = (String)userInfo.get ("USER_ID");`
	
	Refer to USER_INFO table below for values inside this HashMap.

SYS_PARAMS	**Hash Map** of all System Parameters passed to the Report. KEY of each element is the name of the system parameter name and VALUE is the string of parameter value.
USER_PARAMS	**Hash Map** of all user parameters passed to the report. KEY of each element is the parameter name and VALUE is the string of parameter value.

beforeReportExecution

```
public Object beforeReportExecution() throws Exception
{
      return new HashMap();
}
```

Place the logical code in above method.

AFTER REPORT EXECUTION

To set the details after report get executed. . This event is triggered after Report Execution.

The following method of the class will be called in this event.

void afterReportExecution(Object eventInfo)

This method is to provide the information related to the report execution to the callback class. It takes an Object eventInfo as parameter which is generally a HashMap object that store as a key, String that specify the property and as a value, a String or HashMap that specifies the customizations on that property.

The Hash Map has either strings or Hash Maps in turn as values. The keys in the hash maps are the key words mentioned below or parameters passed to the report.

The HashMap contains following values.

Key	Value
REPORT_ID: String	The REPORT_ID of the executed report. For example: `String reportID = (String) eventInfo.get ("REPORT_ID");`
CATEGORY_ID: String	The CATEGORY_ID of the executed report.
REPORT_OID: String	The REPORT_OID (Report output ID) of the executed report.
SAVED_OUTPUT_ID: String	The output id of saved report. Useful for referencing a saved report for Viewing.
TIME_SPENT:	Time spent in execution of report in milliseconds.
OUTPUT_TIME:	Time stamp of execution completion in Java time stamp milliseconds.
FORMAT: String	Output format requested such as PDF, HTM etc. The saved report can further be viewed in any other format irrespective to this format.

Key	Value

USER_INFO	**Hash Map** of USER_INFO.
	
	`HashMap userInfo = (HashMap) eventInfo.get("USER_INFO") ;`
	`String appID = (String)userInfo.get ("USER_ID");`
	
	Refer to USER_INFO table below for values inside this HashMap.

PARAMS_INFO	**Hash Map** of PARAMS_INFO. Refer to PARAMS_INFO table below.
CONNECTION	**Java.sql.connection** class object : The connection, which was used to execute the report or requested in the getEventInfo call. The connection is in opened state and you have to keep it in the same state by end of the call. This connection is provided from the connection pool of Intellicus database connections.
OPERATION_TYPE	The operation performed on the report. **SAVE** = Report was requested for saving by UI or by scheduler. In this case, SAVED_OUTPUT_ID is also available in this HashMap for future use. **EXEC =** The report was requested for viewing on the UI. In this case, SAVE_OUTPUT_ID will be blank or null. **PRINT =** The report was requested for printing on the UI or on the server side. In this case, SAVE_OUTPUT_ID will be blank or null.
ERROR_CODE	NULL or Blank if no error occurred. Contains the error code / number if any error occurs in the execution of the report.
ERROR_MESSAGE	Error message if any error occurs.
ERROR_ROOT_CAUSE	Descriptive error message in case an error occurred.
SYS_PARAMS	**Hash-Map** of all System Parameters passed to the Report. KEY of each element is the name of the system parameter name and VALUE is the string of parameter value.

Key	Value
USER_PARAMS	**Hash Map** of all user parameters passed to the report. KEY of each element is the parameter name and VALUE is the string of parameter value.

afterReportExecution

```
public void afterReportExecution(Object eventInfo) throws
Exception
{
  //Get the saved report id in case a report is being saved
  String execType = (String) eventInfoMap.get("OPERATION TYPE");
  if (execType.equals("SAVE")) // operation type was SAVE
  {
        String errorCode = (String) eventInfoMap.get
                                          ("ERROR_CODE");
        if (errorCode != null )  // means report was successfully
                           // executed and saved !
        {
              String reportSaveID = (String) eventInfoMap.get
                                          ("SAVED_OUTPUT_ID");
        }
  }
}
```

Place the logical code in above method.

BEFORE URL ASSIGN

To get the modified URL. This event will be triggered if the report has hyper-link set for any of its corresponding column.

Passed URL can be:

- Absolute URL
- DrillDown Report.

The following method of the class will be called in this event.

Object beforeURLAssign(Object eventInfo)

This method is used by the Intellicus Report Server to provide a mechanism of doing some pre-processing on the URL being embedded within the report.

This method is to provide the information related to the report execution to the callback class. It takes an Object eventInfo as parameter which is generally a

HashMap object that store as a key, String that specify the property and as a value, a String or HashMap that specifies the customizations on that property.

The HashMap contains following values.

Key	**Value**
URL : String	The drilldown URL
REPORT_ID: String	The REPORT_ID of the executed report. For eg. `String reportID = (String) eventInfo.get ("REPORT_ID");`
CATEGORY_ID: String	The CATEGORY_ID of the executed report.
REPORT_OID: String	The REPORT_OID (Report output ID) of the executed report.
SAVED_OUTPUT_ID: String	The output id of saved report. Useful for referencing a saved report for Viewing.
FORMAT: String	String: Output format requested such as PDF , HTM etc. The saved report can further be viewed in any other format irrespective to this format.
TIME_SPENT: long	Time spent in execution of report in milliseconds.
OUTPUT_TIME: long	Time stamp of execution completion in Java time stamp milliseconds.

USER_INFO	**Hash Map** of USER_INFO.
	`HashMap userInfo = (HashMap) eventInfo.get("USER_INFO") ;`
	`String appID = (String)userInfo.get ("USER_ID");`
	Refer to USER_INFO table below for values inside this HashMap.

PARAMS_INFO	Hash Map of PARAMS_INFO. Refer to PARAMS_INFO table below.
OPERATION_TYPE	The operation performed on the report. **SAVE** = Report was requested for saving by UI or by scheduler. In this case, SAVED_OUTPUT_ID is also available in this HashMap for future use. **EXEC =** The report was requested for viewing on the UI. In this case, SAVE_OUTPUT_ID will be blank or null. **PRINT =** The report was requested for printing on the UI or on the server side. In this case, SAVE_OUTPUT_ID will be blank or null.

User Info Hash Map

Key	Value
USER_ID	USER ID passed to Report Server for execution of report.
PASSWORD	Password passed to Report Server for execution of report.
ORG_ID	ORGID passed to Report Server for execution of report.
SESSION_ID	Session id.
SECURITY_DESCRIPTOR	Security Descriptor string.
CUSTOMER_ID	Customer id, for Service Provider deployments.
LOCATION	Location.
LOCALE	Locale setting of browser.
TIME_STAMP	Time of request in Java time stamp milliseconds.
DB_NAME	Data base name.
ROLES	Roles granted to the user.
CONNECTION_NAME	Connection name requested to use for the report.

Params Info Hash Map

Key	Value
SYS_PARAMS	**Hash Map** of all System Parameters passed to the Report. KEY of each element is the name of the system parameter name and VALUE is the string of parameter value.
USER_PARAMS	**Hash Map** of all user parameters passed to the report. KEY of each element is the parameter name and VALUE is the string of parameter value.

This code will be called for all those reports having Hyper-link set corresponding in its column. To modify the URL only for specific Reports, implementation code needs to be modified as per requirement.

beforeURLAssign

```
public Object beforeURLAssign(Object eventInfoMap)
{
  //This HashMap will return an assigned URL
  String url = (String)((HashMap)eventInfoMap).get("URL");

  //This HashMap will return a UserInfo HashMap which contains
  user_id and password of Logged in User.
  HashMap userInfo =
  (HashMap)((HashMap)eventInfoMap).get("USER_INFO");

  //Get User id from userInfo hashMap by passing USER_ID key.

  //This is when the passed URL is for drilldown to any
  Intellicus report.
  if(url.contains("DRILLDOWN")){
        //Write your Custom code in case of drill down Reports
  }
  //This is when the passed URL is an absolute URL.
  else{
        //Url String contains URL:~TARGET=0:SRC= additional value
        in a URL.
        //Below line of code is used to get Initial prefix of URL.
        String initialURlPrefix =

        url.substring(0,url.indexOf("SRC=")+"SRC=".length());
        //Below line of code is used to get orginal URL.
```

```
        String originalURL =
        url.substring(url.indexOf("SRC=")+"SRC=".length());
        //Below condition check wheter given url contains any
              parameter  or not.
        //Custom code to get modified url
        originalURL = getURL(originalURL);
        //Append intial prefix of URL and original Url to form a
              new   URL.
        url = initialURlPrefix + originalURL;
  }
  //create a HashMap Object
  HashMap hmapModifiedURL = new HashMap();
  //put modified url in a created HashMap
  hmapModifiedURL.put("URL", url);
  //Return above HashMap to calling function.
  return hmapModifiedURL;
}
```

When the passed URL is for drilldown to any Intellicus report.

```
if(url.contains("DRILLDOWN")){
  //Write your Custom code here
}
```

When the passed URL is an absolute URL.

```
else{
  //Write your Custom code here
}
```

Place the logical code in above method

BEFORE PARAMETERS INITIALIZATION

This method is used by Intellicus Report Server to provide a mechanism of doing some pre-processing on the User Parameters.

The following method of the class will be called in this event.

public void beforeParamsInitialization(java.lang.Object eventInfo) throws java.lang.Exception

This method takes a HashMap Object eventInfo that store as a key, String that specify the property and as a value, a String that specifies the customizations on that property. The HashMap contains following values.

The HashMap contains following values.

Key	Value
OPERATION_TYPE: String	The operation performed on the report like SAVE,EXEC,PRINT etc.
OUTPUT_TIME : long	Time stamp of execution completion in Java time stamp milliseconds.

USER_INFO	**Hash Map** of USER_INFO.
	`HashMap userInfo = (HashMap) eventInfo.get("USER_INFO") ;`
	`String appID = (String)userInfo.get ("USER_ID");`
	Refer to USER_INFO table below for values inside this HashMap.

USER_PARAMS: HashMap	Hash Map of all user parameters passed to the report. KEY of each element is the parameter name and VALUE is the string of parameter value.

User Info Hash Map

Key	Value
USERINFO_USERID	USER ID passed to Report Server for execution of report.
USERINFO_SESSIONID	Session id.
USERINFO_ORGID	ORGID passed to Report Server for execution of report.
USERINFO_SECURITY_DESCRIPTOR	Security Descriptor string.
USERINFO_CUSTOMERID	Customer id, for Service Provider deployments.
USERINFO_LOCATION	Location.
USERINFO_LOCALE	Locale setting of browser.
USERINFO_TIME_STAMP	Time of request in Java time stamp milliseconds.
USERINFO_DB_NAME	Data base name.
USERINFO_ROLES	Roles granted to the user.
USERINFO_CONNECTION_NAME	Connection name requested to use for the report.

This code will be called to provide a mechanism of doing some pre-processing on the parameters

beforeURLAssign

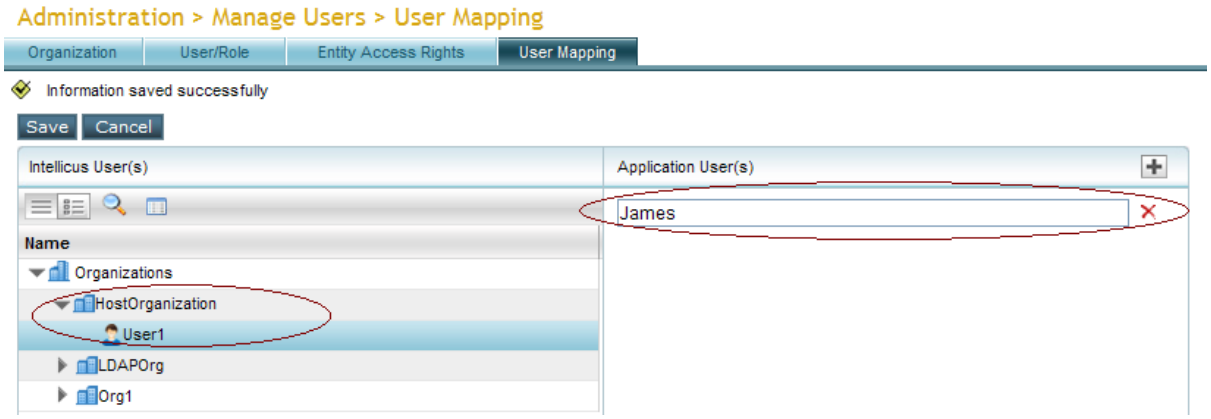
```
public Object beforeParamsInitialization(Object eventInfo)
{
//The HashMap contains following values.
//      1)    OPERATION_TYPE String => The operation performed on
the report like SAVE,EXEC,PRINT etc
//      2)    OUTPUT_TIME long => Time stamp of execution
completion in Java time stamp milliseconds.
//      3)    USER_INFO HashMap => Hash Map of USER_INFO which
contains following values.
//      4)    USER_PARAMS HashMap => Hash Map of all user
parameters passed to the report.
//            KEY of each element is the parameter name and VALUE
is the string of parameter value.
HashMap paramsInfoMap = (HashMap) eventInfo;

//Get USER_PARAMS HashMap
HashMap userParamsMap = (HashMap)
paramsInfoMap.get("USER_PARAMS");
System.out.println(userParamsMap.keySet());

//Here sample code assumes that USER_PARAMS hashmap contains 2
user parameter with name START_TIME & END_TIME
//which host application wants to update before Params
Initialization

//Get START_TIME, a sample user parameter, from userParamsMap
String StartParam = (String)userParamsMap.get("START_TIME");
//If StartParam is not null then update START_TIME
if(StartParam != null)  {
  // Update START_TIME in userParamsMap
  // Here host application will replace START_TIME parameter with
actual time after proper calculation
  userParamsMap.put("START_TIME", new Date().toString());
  System.out.println("Parameter START_TIME found");
}

//Get END_TIME, a sample user parameter,user parameter from
userParamsMap
String EndParam = (String)userParamsMap.get("END_TIME");
```

```
//If EndParam is not null then update END_TIME
if(EndParam != null)    {
  // Update END_TIME in userParamsMap
  // Here host application will replace END_TIME parameter with
actual time after proper calculation
  userParamsMap.put("END_TIME", new Date().toString());
  System.out.println("Parameter END_TIME found");
}
```

Place the logical code in above method

User Mapping

Application user needs to be mapped with Intellicus user.



Connection Events

Intellicus provides a class "ConnectionEvents". This class provides the below given methods. Intellicus integrator has to write a java class, which must extend this class for Intellicus to call a sequence of methods.

Configuration

For implementing **Connection Events**, **EVENTSHANDLER TYPE** should be: CONNECTIONEVENTS.

Following code shows the XML entry having configuration of Connection Events.

```
<EVENTSHANDLER TYPE=" CONNECTIONEVENTS">
  <CALLBACK CALLTYPE="1">
       <IMPLEMENTER TYPE="1">
            <ATTRS TYPE="1">
                 <ATTR NAME="PATH">
                      <VALUE>com.mypackage.myclass</VALUE>
                 </ATTR>
            </ATTRS>
       </IMPLEMENTER>
```

```
  </CALLBACK>
</ EVENTSHANDLER>
```

During callback events process Intellicus Report Server will call these methods:

Object beforeConnectionGet(Object connInfo): Get callback event information before the connection of any operation is being get from the connection pool. Gather these information from the java class using HashMap as object.

1. **Object afterConnectionGet(Object connInfo):** Set the callback event Infomation after the connection of any operation is being get from connection pool Provide these information to the java class using HashMap as object.

2. **Object beforeConnectionSubmit(Object connInfo):** Get callback event information before the connection of any operation is being submitted back to the connection pool. Gather these information from the java class using HashMap as object.

Sample Implementation code

```
import com.impetus.interaj.callback.ConnectionEvents;

public class SampleConnectionCallbackEvent extends
ConnectionEvents {

}
```

BEFORE CONNECTION GET

To get the Connection Name along with other parameters before a connection is get from connection pool.

The following method of the class will be called in this event.

Object beforeConnectionGet(Object connInfo)

Intellicus call backs an event before getting a connection from connection pool. There might be situation that clients want to change connection on which a particular operation is to be performed. So here Intellicus first raises an event (i.e. beforeConnectionGet) before getting a connection from connection pool. It provides the connection name in the callback code. The callback code may change the connection name for particular operation. Intellicus should receive the updated connection name and should also use the updated connection for the execution of an operation.

- This callback event executes each time Intellicus tries to get a connection from Connection Pool.
- This callback event happens for non-repository connections only.
- This event can be raised for the execution of following components.

- Chart Execution
- Crosstab Execution
- Main Report Execution
- Sub Report Execution

Event Info Hash Map

Key	Value
CONNECTION_NAME	Name of a Connection.
OBJECT_TYPE	Operation is performed for which type of object: Chart, Crosstab etc.
OBJECT_ID	Object Id.
IS_DEFAULT	Connection is default or not.
IS_REPOSITORY	Connection is repository connection or not.
USER_INFO	Hash Map of USER_INFO. Refer to USER_INFO table below for values inside this HashMap.
SYS_PARAMS	Hash Map of all System Parameters passed to the Report. KEY of each element is the name of the system parameter name and VALUE is the string of parameter value.
USER_PARAMS	Hash Map of all user parameters passed to the report. KEY of each element is the parameter name and VALUE is the string of parameter value.

beforeConnectionGet

```
public Object beforeConnectionGet(Object beforeConnInfo){

  String conName =
        (String)((HashMap)beforeConnInfo).get("CONNECTION_NAME");
  String isDefault =
        (String)((HashMap)beforeConnInfo).get("IS_DEFAULT");
  String isRepository =
        (String)((HashMap)beforeConnInfo).get("IS_REPOSITORY");

  System.out.println("Connection Name :  "+conName);
  System.out.println("Is Connection Default : "+isDefault);
  System.out.println("Is Connection Repository : "+isRepository);

  return beforeConnInfo;
}
```

Place the logical code in above method.

AFTER CONNECTION GET

To get the Connection Details if connections get successfully from the connection pool.

The following method of the class will be called in this event.

Object afterConnectionGet(Object connInfo)

Intellicus can callback an event when it gets any connection from a connection pool. Intellicus Application supports pooling of data sources connections. A connection is acquired at the time of report execution/Chart Query Execution/Cross tab Query Execution/Sub Report Query Execution etc. A client may want to change some data source property while getting the connection. So Intellicus can callback an event (i.e. afterConnectionGet) on a callback instance whenever a connection is acquired from the pool. It can also provide the connection object to the callback event. The call back code may change Connection Properties so that Intellicus receives the updated connection object to use further.

- This callback event executes each time a connection is acquired.
- This call back happens for non-repository connections only.
- This event is raised for the execution of following components.
    - Chart Execution
    - Crosstab Execution
    - Main Report Execution
    - Sub Report Execution

Event Info Hash Map

Key	Value
CONNECTION	Entire java.sql.connection object.
CONNECTION_NAME	Name of a Connection.
IS_DEFAULT	Connection is default or not.
IS_REPOSITORY	Connection is repository connection or not.
USER_INFO	**Hash Map** of USER_INFO. Refer to USER_INFO table below for values inside this HashMap.
SYS_PARAMS	**Hash Map** of all System Parameters passed to the Report. KEY of each element is the name of the system parameter name and VALUE is the string of parameter value.
USER_PARAMS	**Hash Map** of all user parameters passed to the report. KEY of each element is the parameter name and VALUE is the string of parameter value.

afterConnectionGet

```
public Object afterConnectionGet(Object afterConnInfo){

  Connection c =
        (Connection)((HashMap)afterConnInfo).get("CONNECTION");
  System.out.println("Connection "+c.toString());

  String conName =
  (String)((HashMap)afterConnInfo).get("CONNECTION_NAME");
  String isDefault =
        (String)((HashMap)afterConnInfo).get("IS_DEFAULT");
  String isRepository =
        (String)((HashMap)afterConnInfo).get("IS_REPOSITORY");
  HashMap userDetail =
        (HashMap)((HashMap)afterConnInfo).get("USER_INFO");
  String userId =
        (String)((HashMap)userDetail).get("USERINFO_USERID");
  String orgId =
        (String)((HashMap)userDetail).get("USERINFO_ORGID");

  System.out.println("Conn Name : "+conName);
  System.out.println("Is Conn Default : "+isDefault);
  System.out.println("Is Conn Repository : "+isRepository);
  System.out.println("User Id : "+userId);
  System.out.println("Org Id : "+orgId);

  return afterConnInfo;
}
```

Place the logical code in above method.

BEFORE CONNECTION SUBMIT

To get the Connection Details before connection is submitted back to the connection pool.

The following method of the class will be called in this event.

Object beforeConnectionSubmit(Object connInfo)

Intellicus supports callback event for the operation of submitting any connection back to connection pool. As Intellicus gives access to set some properties while getting any connection from connection pool, so clients may need to reset those properties that are being set at the time of connection retrieval. Intellicus raises a callback event (i.e. beforeConnectionSubmit) on a callback instance. It also provides the connection object to the callback event. So in the call back code

clients may reset Connection Properties. And Intellicus should receive the updated connection object from callback code of client.

- This call back event executes each time a connection is submitted back to connection pool.
- This call back happens for non-repository connections only.
- This event can be raised for the execution of following components.
  - Chart Execution
  - Crosstab Execution
  - Main Report Execution
  - Sub Report Execution

Event Info Hash Map

Key	Value
CONNECTION	Entire java.sql.connection object.
PROVIDER	Database Provider.
CONNECTION_NAME	Name of a Connection.
REPORT_ID	Report Id.
REPORT_NAME	Report Name.
CATEGORY_ID	Category Id.
CATEGORY_NAME	Category Name.
OBJECT_TYPE	Operation is performed for which type of object: Chart, Crosstab etc.
OBJECT_ID	Object Id.
USER_PARAMS	User parameters.
IS_DEFAULT	Connection is default or not.
IS_REPOSITORY	Connection is repository connection or not.
USER_INFO	Hash Map of USER_INFO. Refer to USER_INFO table below for values inside this HashMap.

USER INFO HASHMAP

Key	Value
USERINFO_USERID	USER ID passed to Report Server for execution of report.
USERINFO_ORGID	ORGID passed to Report Server for execution of report.
USERINFO_SESSIONID	Session id.
USERINFO_SECURITY_DESCRIPTOR	Security Descriptor string.

Key	Value
USERINFO_CUSTOMERID	Customer id, for Service Provider deployments.
USERINFO_LOCATION	Location.
USERINFO_LOCALE	Locale setting of browser.
USERINFO_TIMESTAMP	Time of request in Java time stamp milliseconds.
USERINFO_DBNAME	String: Data base name.
USERINFO_ROLES	Roles granted to the user.
USERINFO_CONNECTION_NAME	Connection name requested to use for the report.

beforeConnectionSubmit

```
public Object beforeConnectionSubmit(Object beforeConnInfo){

  String conName =
        (String)((HashMap)beforeConnInfo).get("CONNECTION_NAME");
  String isDefault =
        (String)((HashMap)beforeConnInfo).get("IS_DEFAULT");
  String isRepository =
        (String)((HashMap)beforeConnInfo).get("IS_REPOSITORY");
  HashMap userDetail =
        (HashMap)((HashMap)beforeConnInfo).get("USER_INFO");
  String userId =
        (String)((HashMap)userDetail).get("USERINFO_USERID");
  String orgId =
        (String)((HashMap)userDetail).get("USERINFO_ORGID");

  System.out.println("Conn Name :   "+conName);
  System.out.println("Is Conn Default : "+isDefault);
  System.out.println("Is Conn Repository : "+isRepository);
  System.out.println("User Id : "+userId);
  System.out.println("Org Id : "+orgId);

  return beforeConnInfo;
  }
```

Place the logical code in above method.

UMM Events

Intellicus facilitates callback events mechanism for User Management. These events will be triggered whne any User management activity like, User creation, Role creation, Organization creation or User modification takes place.

Configuration

For implementing **UMM Events**, **EVENTSHANDLER TYPE** should be: UMMEVENTS.

Following code shows the XML entry having configuration of UMM Events.

```
<EVENTSHANDLER TYPE=”UMMEVENTS”>
  <CALLBACK CALLTYPE="1">
        <IMPLEMENTER TYPE="1">
              <ATTRS TYPE="1">
                    <ATTR NAME="PATH”>
                          <VALUE>com.mypackage.myclass</VALUE>
                    </ATTR>
              </ATTRS>
        </IMPLEMENTER>
  </CALLBACK>
</ EVENTSHANDLER>
```

During callback events process Intellicus Report Server will call these methods:

1. **void afterOrganizationCreate(Object orgInfo):** Provide the callback event Infomation after the new Organization is created. Provide these information to the java class using HashMap as object.

2. **void afterRoleCreate(Object roleInfo):** Provide the callback event Infomation after the new Role is created. Provide this information to the java class using HashMap as object.

3. **void beforeUserCreate(Object userInfo):** Provide the callback event Infomation before the new User is created. Provide these information to the java class using HashMap as object.

4. **void afterUserCreate(Object userInfo):** Provide the callback event Infomation after the new User is created. Provide this information to the java class using HashMap as object.

5. **void beforeUserModify(Object userInfo):** Provide the callback event Infomation before any User is modified. Provide these information to the java class using HashMap as object.

6. **void afterUserModify(Object userModInfo):** Provide the callback event Infomation after the existing User is modified. Provide this information to the java class using HashMap as object.

Sample Implementation code

```
import com.impetus.interaj.callback.UMMEvents;

public class SampleUMMCallbackEvent extends UMMvents {

}
```

AFTER ORGANIZATION CREATE

To get the Organization details when new organization is being added to Intellicus. This event is triggered After Organization is created.

The following method of the class will be called in this event.

void afterOrganizationCreate(Object orgInfo)

Intellicus supports callback event for new organization creation in Application. In Intellicus, users are allowed to create new organization. So at the time new organization is created, Intellicus raises a callback event (i.e. afterOrganizationCreate) on the call back instance. It also provides the organization related information to the callback event. So at the callback code clients may view all organization properties.

This callback event executes each time a new organization is being created in Intellicus.

Event Info Hash Map

Key	Value
ORGANIZATION_ID	Organization Id.
USER_INFO	**Hash Map** of USER_INFO. Refer to USER_INFO table below for values inside this HashMap.

USER_INFO HashMap

Key	Value
USERINFO_USERID	USER ID passed to Report Server for execution of report.
USERINFO_ORGID	ORGID passed to Report Server for execution of report.
USERINFO_SESSIONID	Session id.
USERINFO_SECURITY_DESCRIPTOR	Security Descriptor string.
USERINFO_CUSTOMERID	Customer id, for Service Provider deployments.

Key	Value
USERINFO_LOCATION	Location.
USERINFO_LOCALE	Locale setting of browser.
USERINFO_TIMESTAMP	Time of request in Java time stamp milliseconds.
USERINFO_DBNAME	String: Data base name.
USERINFO_ROLES	Roles granted to the user.
USERINFO_CONNECTION_NAME	Connection name requested to use for the report.

afterOranizationCreate

```
public void afterOrganizationCreate(Object orgInfo){
  String orgId =
        (String)((HashMap)orgInfo).get("ORGANIZATION_ID");
  System.out.println("Organization Id : "+orgId);

  HashMap userDetail = (HashMap)
        ((HashMap)orgInfo).get("USER_INFO");
  System.out.println("User Id :
              "+userDetail.get("USERINFO_USERID"));
  System.out.println("Organization Id :
              "+userDetail.get("USERINFO_ORGID"));

}
```

Place the logical code in above method.

AFTER ROLE CREATE

To get Role details when new Role is being added to Intellicus. This event is triggered after new role is created.

The following method of the class will be called in this event.

void afterRoleCreate(Object roleInfo)

Intellicus supports callback event for new role creation in Application. In Intellicus, users are allowed to create new roles. So at the time new role is created, Intellicus raises a callback event (i.e. afterRoleCreate) on a callback instance. It also provides the Role related information to the callback event. So at the callback code clients may view all Role related properties.

This callback event executes each time, a new Role is being created in Intellicus.

Event Info Hash Map

Key	Value
ROLE_ID	Role Id.
ORGANIZATION_ID	Organization Id in which role is being added.

USER INFO HASHMAP

Key	Value
USERINFO_USERID	USER ID passed to Report Server for execution of report.
USERINFO_ORGID	ORGID passed to Report Server for execution of report.
USERINFO_SESSIONID	Session id.
USERINFO_SECURITY_DESCRIPTOR	Security Descriptor string.
USERINFO_CUSTOMERID	Customer id, for Service Provider deployments.
USERINFO_LOCATION	Location.
USERINFO_LOCALE	Locale setting of browser.
USERINFO_TIMESTAMP	Time of request in Java time stamp milliseconds.
USERINFO_DBNAME	String: Data base name.
USERINFO_ROLES	Roles granted to the user.
USERINFO_CONNECTION_NAME	Connection name requested to use for the report.

afterRoleCreate

```
public void afterRoleCreate(Object roleInfo){
  String roleId = (String)((HashMap)roleInfo).get("ROLE_ID");
  String orgId =
        (String)((HashMap)roleInfo).get("ORGANIZATION_ID");
  String rStatus= (String)((HashMap)roleInfo).get("ROLE_STATUS");
  String rAdmin =(String)((HashMap)roleInfo).get("ROLE_ADMIN");
  String sysPriveleges =
        (String)((HashMap)roleInfo).get("SYSTEM_PRIVILEGES");

  System.out.println("Role -- Role Id"+roleId);
  System.out.println("Role -- Organization Id"+orgId);
  System.out.println("Role -- Role Status"+ rStatus);
  System.out.println("Role -- Role Admin"+ rAdmin);
  System.out.println("Role -- System Priveleges"+sysPriveleges);
}
```

Place the logical code in above method.

BEFORE USER CREATE

User details when new User is being added to Intellicus. This event is triggered just before the new User is created.

The following method of the class will be called in this event.

void beforeUserCreate(Object userInfo)

Intellicus supports callback event for new user creation in Application. In Intellicus Admins /Super Admins are allowed to create new users. So at the time new user is created, Intellicus raises a callback event (i.e. beforeUserCreate) on the call back instance. It also provides the User related information i.e. User ID and Passowrd to the callback event.

This callback event executes each time before a new User is being created in Intellicus.

UserInfo Hash Map

Key	Value
USER_ID	User Id.
PASSWORD	User Password.

beforeUserCreate

```
//This method will be used to view User Information in callback
code.
//This event will be raised each time before a User is created.
public void beforeUserCreate(Object userInfo){
    System.out.println("inside beforeUserCreate()..............");
    String userId = (String)((HashMap)userInfo).get("USER_ID");
    String password = (String)((HashMap)userInfo).get("PASSWORD");
    System.out.println("User Id---"+userId);
    System.out.println("Passowrd ---"+password);
}
```

Place the logical code in above method.

AFTER USER CREATE

User details when new User is being added to Intellicus. This event is triggered just after new User is created.

The following method of the class will be called in this event.

void afterUserCreate(Object userInfo)

Intellicus supports callback event for new user creation in Application. In Intellicus Admins /Super Admins are allowed to create new users. So at the time

new user is created, Intellicus raises a callback event (i.e. afterUserCreate) on the call back instance. It also provides the User related information to the callback event. So in the call back code clients may view all User related properties or various privileges assigned to user.

This callback event executes each time a new User is being created in Intellicus.

Event Info Hash Map

Key	Value
USER_ID	User Id.
DESCRIPTION	User Description.
USER_STATUS	User Status, Whether the User is Active or suspended.
ORGANIZATION_ID	Organization Id in which User is being added.
PASSWORD	User Password.
IS_FIRST_LOGIN	Is user login for the first time or not?
IS_BALNK_PASSWORD	Is the password blank or not.
PWD_LAST_CHANGED	Date when password changed last time.
IS_SUPER_ADMIN	Is the user is Super Admin or not.
IS_ADMIN	Is the User is Admin or not.
SYSTEM_PRIVILEGES	System Privileges assigned to user.
USER_PREFERENCES	Hash Map of User Preferences set for the user. Refer to USER_PREFERENCES table below for values inside this HashMap.
USER_INFO	**Hash Map** of USER_INFO. Refer to USER_INFO table below for values inside this HashMap.

USER PREFERENCES HASHMAP

Key	Value
DEFAULT_FORMAT	Default Format set for the user.
LANG	Default Language set for the user.
THEME	Default Theme set for the user.
DEFAULT_DASHBOARD	Default Dashboard set for the user.
SHOW_RECENT_REPORTS	Whether Recent Reports to be shown or not.
SHOW_RECENT_REP_PUBLISHED	Whether Recent Published Reports to be shown or not.
REPORT_CACHE_SIZE	Report Cache Size.
EMAIL	Email Id of the newly added user.
APPID	Id of the user who is setting user preferences.
ORGID	Organization Id of the user who is setting user preferences.
DEFAULT_CONNECTION	Default Connection set for the user.
RUNTIME_USER_ID	Runtime User Id of the user.
RUNTIME_PASSWORD	Runtime Password of the user.
TEMPLATE_NAME	Template name set for the user.
USE_EMAIL_FOR	Returns ENUMS values for which

	Email has to be sent. (0=WORKFLOW_APPROVAL, 1=ASYNCHRONOUS_REPORT_ COMPLETION, 2=MEMORY_TRIGGER, 3=EXEC_REJECT)

USER INFO HASHMAP

Key	Value
USERINFO_USERID	USER ID passed to Report Server for execution of report.
USERINFO_ORGID	ORGID passed to Report Server for execution of report.
USERINFO_SESSIONID	Session id.
USERINFO_SECURITY_DESCRIPTOR	Security Descriptor string.
USERINFO_CUSTOMERID	Customer id, for Service Provider deployments.
USERINFO_LOCATION	Location.
USERINFO_LOCALE	Locale setting of browser.
USERINFO_TIMESTAMP	Time of request in Java time stamp milliseconds.
USERINFO_DBNAME	String: Data base name.
USERINFO_ROLES	Roles granted to the user.
USERINFO_CONNECTION_NAME	Connection name requested to use for the report.

afterUserCreate

```
public void afterUserCreate(Object userInfo){
  System.out.println("inside afterUserCreate()..............");

  String userId = (String)((HashMap)userInfo).get("USER_ID");
  String orgId = (String)((HashMap)userInfo).get
                                    ("ORGANIZATION_ID");
  String userStatus = (String)((HashMap)userInfo).get
                                    ("USER_STATUS");
  String isSuperAdmin = (String)((HashMap)userInfo).get
                                    ("IS_SUPER_ADMIN");
  String isAdmin = (String)((HashMap)userInfo).get("IS_ADMIN");

  HashMap userPref = (HashMap) ((HashMap)userInfo).get
                                    ("USER_PREFERENCES");
```

```
    String defFormat =     (String)((HashMap)userPref).get
                                    ("DEFAULT_FORMAT");
    String lang = (String)((HashMap)userPref).get("LANG");
    String theme = (String)((HashMap)userPref).get("THEME");
    String templateName = (String)((HashMap)userPref).get
                                    ("TEMPLATE_NAME");
    String repCacheSize = (String)((HashMap)userPref).get
                                    ("REPORT_CACHE_SIZE");
    String defDashboard = (String)((HashMap)userPref).get
                                    ("DEFAULT_DASHBOARD");
    System.out.println("User Id---"+userId);
    System.out.println("Org Id---"+orgId);
    System.out.println("User Status---"+userStatus);
    System.out.println("Is Super Admin---"+isSuperAdmin);
    System.out.println("Is Admin---"+isAdmin);
    System.out.println("defFormat---"+defFormat);
    System.out.println("language---"+lang);
    System.out.println("Theme---"+theme);
    System.out.println("Template Name---"+templateName);
    System.out.println("Report Cache Size---"+repCacheSize);
    System.out.println("Default Dashboard ---"+defDashboard);
  }
```

Place the logical code in above method.

BEFORE USER MODIFY

User details when existing User is being modified in Intellicus. This event is triggered just before modifying the User details.

The following method of the class will be called in this event.

void beforeUserModify(Object userInfo)

Whenever any existing user is modified in Intellicus, in the call back code, clients will get the details of the user getting modified. User details will be available in call back as UserInfo parameter which is a hashmap that contains User Information.

The HashMap contains following values.

Key	Value
USER_ID	User Id.
PASSWORD	User Password.
PWD_LAST_CHANGED	Date when password changed last time.
IS_UPDATE_PASSWORD	True if password is changed.

beforeUserModify

```
//This method will be used to view User Information in callback code.
//This event will be raised each time before any existing User is
modified.
public void beforeUserModify(Object userInfo){
      System.out.println("inside beforeUserModify().........");
      String userId= (String)((HashMap)userInfo).get("USER_ID");
      String password= (String)((HashMap)userInfo).get("PASSWORD");
      String date=(String)
((HashMap)userInfo).get("PWD_LAST_CHANGED");
      String isPsswordUpdated =
(String)((HashMap)userInfo).get("IS_UPDATE_PASSWORD");

      System.out.println("User Id---"+userId);
      System.out.println("Passowrd ---"+password);
      System.out.println("Passowrd Changed ---"+isPsswordUpdated);
      System.out.println("Password Last Changed ---"+date);
}
```

Place the logical code in above method.

AFTER USER MODIFY

User details when existing User is being modified in Intellicus. This event is triggered just after modifying the User details.

The following method of the class will be called in this event.

void afterUserModify(Object userInfo)

Whenever any existing user is modified in Intellicus, in the call back code, clients will get all the details of the user modified. User details will be available in call back.

Key	Value
USER_ID	User Id.
DESCRIPTION	User Description.
USER_STATUS	User Status, Whether the User is Active or suspended.
ORGANIZATION_ID	Organization Id of the modified user.
PASSWORD	User Password.
IS_FIRST_LOGIN	Is user login for the first time or not?
IS_BALNK_PASSWORD	Is the password blank or not.
PWD_LAST_CHANGED	Date when password changed last time.
IS_SUPER_ADMIN	Is the user is Super Admin or not.

Key	Value
IS_ADMIN	Is the User is Admin or not.
SYSTEM_PRIVILEGES	System Privileges assigned to user.
USER_PREFERENCES	Hash Map of User Preferences set for the user. Refer to USER_PREFERENCES table below for values inside this HashMap.
USER_INFO	**Hash Map** of USER_INFO. Refer to USER_INFO table below for values inside this HashMap.

USER PREFERENCES HASHMAP

Key	Value
DEFAULT_FORMAT	Default Format set for the user.
LANG	Default Language set for the user.
THEME	Default Theme set for the user.
DEFAULT_DASHBOARD	Default Dashboard set for the user.
SHOW_RECENT_REPORTS	Whether Recent Reports to be shown or not.
SHOW_RECENT_REP_PUBLISHED	Whether Recent Published Reports to be shown or not.
REPORT_CACHE_SIZE	Report Cache Size.
EMAIL	Email Id of a user.
APPID	Id of the user who is setting user preferences.
ORGID	Organization id of the user who is setting user preferences.
DEFAULT_CONNECTION	Default Connection set for the user.
RUNTIME_USER_ID	Runtime User Id of the user.
RUNTIME_PASSWORD	Runtime Password of the user.
TEMPLATE_NAME	Template name set for the user.
USE_EMAIL_FOR	Returns ENUMS values for which Email has to be sent. (0=WORKFLOW_APPROVAL, 1=ASYNCHRONOUS_REPORT_ COMPLETION, 2=MEMORY_TRIGGER, 3=EXEC_REJECT)

USER INFO HASHMAP

Key	Value
USERINFO_USERID	USER ID passed to Report Server for execution of report.
USERINFO_ORGID	ORGID passed to Report Server for execution of report.
USERINFO_SESSIONID	Session id.
USERINFO_SECURITY_ DESCRIPTOR	Security Descriptor string.
USERINFO_CUSTOMERI D	Customer id, for Service Provider deployments.
USERINFO_LOCATION	Location.
USERINFO_LOCALE	Locale setting of browser.
USERINFO_TIMESTAMP	Time of request in Java time stamp milliseconds.
USERINFO_DBNAME	String: Data base name.
USERINFO_ROLES	Roles granted to the user.
USERINFO_CONNECTIO N_NAME	Connection name requested to use for the report.

afterUserModify

```
public void afterUserModify(Object modUserInfo)
{
  System.out.println("Inside after User Modified().........");
  String userId = (String)((HashMap)modUserInfo).get("USER_ID");
  String orgId = (String)((HashMap)modUserInfo).get
                                          ("ORGANIZATION_ID");
  String userStatus = (String)((HashMap)modUserInfo).get
                                          ("USER_STATUS");
  String isSuperAdmin = (String)((HashMap)modUserInfo).get
                                          ("IS_SUPER_ADMIN");
  String isAdmin= (String)((HashMap)modUserInfo).get("IS_ADMIN");

  HashMap userPref=(HashMap) ((HashMap)modUserInfo).get
                                          ("USER_PREFERENCES");

  String defFormat = (String)((HashMap)userPref).get
                                          ("DEFAULT_FORMAT");
```

```
  String lang = (String)((HashMap)userPref).get("LANG");
  String theme = (String)((HashMap)userPref).get("THEME");
  String templateName = (String)((HashMap)userPref).get
                                        ("TEMPLATE_NAME");
  String repCacheSize = (String)((HashMap)userPref).get
                                        ("REPORT_CACHE_SIZE");
  String defDashboard = (String)((HashMap)userPref).get
                                        ("DEFAULT_DASHBOARD");

  System.out.println("User Id---"+userId);
  System.out.println("Org Id---"+orgId);
  System.out.println("User Status---"+userStatus);
  System.out.println("Is Super Admin---"+isSuperAdmin);
  System.out.println("Is Admin---"+isAdmin);
  System.out.println("defFormat---"+defFormat);
  System.out.println("language---"+lang);
  System.out.println("Theme---"+theme);
  System.out.println("Template Name---"+templateName);
  System.out.println("Report Cache Size---"+repCacheSize);
  System.out.println("Default Dashboard ---"+defDashboard);
}
```

Place the logical code in above method.

REPORTMGMTEVENTS

Intellicus facilitates Auditing callback events mechanism for Report Management Operations by "calling your code" system. For this, Intellicus provides a class "ReportMgmtEvents".

Configuration

For implementing **ReportMgmtEvents**, **EVENTSHANDLER TYPE** should be: REPORTMGMTEVENTS.

Following code shows the XML entry having configuration of UMM Events.

```
<EVENTSHANDLER TYPE=" REPORTMGMTEVENTS">
  <CALLBACK CALLTYPE="1">
        <IMPLEMENTER TYPE="1">
              <ATTRS TYPE="1">
                    <ATTR NAME="PATH">
                          <VALUE>com.mypackage.myclass</VALUE>
                    </ATTR>
              </ATTRS>
        </IMPLEMENTER>
```

```
  </CALLBACK>
</ EVENTSHANDLER>
```

During callback events process Intellicus Report Server will call these methods:

1. void **afterCategoryAdd**(Object categoryInfo)**:** Provides the callback event Infomation after a new Category is added in Intellicus. Provides this information to the java class using HashMap as object.

2. void **afterCategoryModify**(Object categoryInfo)**:** Provides the callback event Infomation after any existing Category is modified in Intellicus. Provides this information to the java class using HashMap as object.

3. void **afterCategoryDelete**(Object categoryInfo)**:** Provides the callback event Infomation after any Category is deleted in Intellicus. Provides this information to the java class using HashMap as object.

4. void **afterReportAdd**(Object reportInfo)**:** Provides the callback event Infomation after a new Report is added in Intellicus. Provides this information to the java class using HashMap as object.

5. void **afterReportModify**(Object reportInfo)**:** Provides the callback event Infomation after any existing Report is modified in Intellicus. Provides this information to the java class using HashMap as object.

6. void **afterReportDelete**(Object reportInfo)**:** Provides the callback event Infomation after any Report is deleted in Intellicus. Provides this information to the java class using HashMap as object.

Sample Implementation code

```
import com.impetus.interaj.callback.ReportMgmtEvents;

public class ReportMgmtCallbackImpl extends AuditEvents {

}
```

AFTER CATEGORY ADD

To get the Category details when new Category is being added to Intellicus. This event is triggered after Category is created.

The following method of the class will be called in this event.

void afterCategoryAdd(java.lang.Object categoryInfo)

Intellicus supports callback event for new Category creation in Application. In Intellicus, users are allowed to create new Categories. So at the time new Category is created, Intellicus raises a callback event (i.e. afterCategoryAdd) on the call back instance. It also provides the Category related information to the callback event. So at the callback code clients may view all Categoryproperties.

This callback event executes each time a new Category is created in Intellicus.

Event Info Hash Map

Key	Value
CATEGORY_OBJECT	Category (Web client's class com.intellica.client.reportutils. Category) object of added category.
USER_INFO	**Hash Map** of USER_INFO. Refer to USER_INFO table below for values inside this HashMap.

USER_INFO HashMap

Key	Value
USERINFO_USERID	USER ID passed to Report Server for execution of report.
USERINFO_ORGID	ORGID passed to Report Server for execution of report.
USERINFO_SESSIONID	Session id.
USERINFO_SECURITY_DESCRIPTOR	Security Descriptor string.
USERINFO_CUSTOMERID	Customer id, for Service Provider deployments.
USERINFO_LOCATION	Location.
USERINFO_LOCALE	Locale setting of browser.
USERINFO_TIMESTAMP	Time of request in Java time stamp milliseconds.
USERINFO_DBNAME	String: Data base name.
USERINFO_ROLES	Roles granted to the user.

USERINFO_CONNECTION_NAME	Connection name requested to use for the report.

afterCategoryAdd

```
public void afterCategoryAdd(Object catInfo){

    System.out.println("Inside afterCategoryAdd()");

    //Get the Category(com.intellica.client.reportutils.Category)
    //object of added category.
    Category catObject = (Category)((HashMap)catInfo).get
                                    ("CATEGORY_OBJECT");

    System.out.println("Category Name : "+catObject.getMenuName());
    System.out.println("CategoryDesc:"+catObject.getDescription());
    System.out.println("Category isPublic =
                    "+catObject.getIsPublic());
    System.out.println("Created Date :
                         "+catObject.getRepositDate());
    System.out.println("Category Access Rights :
                         "+catObject.getAccessRights());
    System.out.println("------- UserInfo details -------");

    //Hash Map of USER_INFO which contains following values.
    HashMap userInfo =
          (HashMap)((HashMap)catInfo).get("USER_INFO");
    System.out.println("USERINFO_USERID ===
          "+((HashMap)userInfo).get("USERINFO_USERID"));
    System.out.println("USERINFO_ORGID ===
          "+((HashMap)userInfo).get("USERINFO_ORGID"));
    System.out.println("USERINFO_SESSIONID ===
          "+((HashMap)userInfo).get("USERINFO_SESSIONID"));
    System.out.println("USERINFO_SD ===
    "+((HashMap)userInfo).get("USERINFO_SECURITY_DESCRIPTOR"));
    System.out.println("USERINFO_CUSTOMERID ===
          "+((HashMap)userInfo).get("USERINFO_CUSTOMERID"));
    System.out.println("USERINFO_LOCATION ===
          "+((HashMap)userInfo).get("USERINFO_LOCATION"));
    System.out.println("USERINFO_LOCALE ===
          "+((HashMap)userInfo).get("USERINFO_LOCALE"));
    System.out.println("USERINFO_DBNAME ===
          "+((HashMap)userInfo).get("USERINFO_DBNAME"));
```

```
        System.out.println("USERINFO_ROLES ===
              "+((HashMap)userInfo).get("USERINFO_ROLES"));
        System.out.println("USERINFO_Conn_Name ===
              "+((HashMap)userInfo).get("USERINFO_CONNECTION_NAME"));
        System.out.println("End Of afterCategoryAdd()! ");
    }
```

Place the logical code in above method.

AFTER CATEGORY MODIFY

This method will be used to Audit modified Category Information in callback code. This event will be raised each time any existing category object is modified.

The following method of the class will be called in this event.

void afterCategoryModify(java.lang.Object categoryInfo)

Event Info Hash Map

Key	Value
CATEGORY_OBJECT	Category (Web client's class com.intellica.client.reportutils. Category) object of modified category.
USER_INFO	**Hash Map** of USER_INFO. Refer to USER_INFO table below for values inside this HashMap.

User Info Hash Map

Key	Value
USERINFO_USERID	USER ID passed to Report Server for execution of report.
USERINFO_ORGID	ORGID passed to Report Server for execution of report.
USERINFO_SESSIONID	Session id.
USERINFO_SECURITY_DESCRIPTOR	Security Descriptor string.
USERINFO_CUSTOMERID	Customer id, for Service Provider deployments.
USERINFO_LOCATION	Location.
USERINFO_LOCALE	Locale setting of browser.
USERINFO_TIMESTAMP	Time of request in Java time stamp milliseconds.
USERINFO_DBNAME	String: Data base name.

USERINFO_ROLES	Roles granted to the user.
USERINFO_CONNECTION_NAME	Connection name requested to use for the report.

afterCategoryModify

```
public void afterCategoryModify(Object catInfo){
    System.out.println("Inside afterCategoryModify() ");

    //Get the Category(com.intellica.client.reportutils.Category)
    //object of modified category.
    Category catObject =
          (Category)((HashMap)catInfo).get("CATEGORY_OBJECT");

    System.out.println("Category Name : "+catObject.getMenuName());
    System.out.println("Category Description :
                    "+catObject.getDescription());
    System.out.println("Category isPublic =
                    "+catObject.getIsPublic());
    System.out.println("Created Date :
                    "+catObject.getReposit Date());
    System.out.println("Category Access Rights :
                    "+catObject.getAccessRights());
    System.out.println("------- UserInfo details -------");

    //Hash Map of USER INFO which contains following values.
    HashMap userInfo =
               (HashMap)((HashMap)catInfo).get("USER INFO");
    System.out.println("USERINFO USERID ===
               "+((HashMap)userInfo).get("USERINFO USERID"));
    System.out.println("USERINFO ORGID ===
               "+((HashMap)userInfo).get("USERINFO ORGID"));
    System.out.println("USERINFO SESSIONID ===
               "+((HashMap)userInfo).get("USERINFO SESSIONID"));
    System.out.println("USERINFO_SD ===
    "+((HashMap)userInfo).get("USERINFO_SECURITY_DESCRIPTOR"));
    System.out.println("USERINFO_CUSTOMERID ===
          "+((HashMap)userInfo).get("USERINFO_CUSTOMERID"));
    System.out.println("USERINFO_LOCATION ===
               "+((HashMap)userInfo).get("USERINFO_LOCATION"));
    System.out.println("USERINFO_LOCALE ===
               "+((HashMap)userInfo).get("USERINFO_LOCALE"));
```

```
    System.out.println("USERINFO_DBNAME ===
                "+((HashMap)userInfo).get("USERINFO_DBNAME"));
    System.out.println("USERINFO_ROLES ===
                "+((HashMap)userInfo).get("USERINFO_ROLES"));
    System.out.println("USERINFO_Conn_Name ===
          "+((HashMap)userInfo).get("USERINFO_CONNECTION_NAME"));
    System.out.println("End Of afterCategoryModify()");
}
```

AFTER CATEGORY DELETE

This method will be used to Audit deleted Category Information in callback code. This event will be raised each time any category object is deleted.

The following method of the class will be called in this event.

void afterCategoryModify(java.lang.Object categoryInfo)

Event Info Hash Map

Key	Value
CATEGORY_OBJECT	Category (Web client's class com.intellica.client.reportutils. Category) object of deleted category.
USER_INFO	**Hash Map** of USER_INFO. Refer to USER_INFO table below for values inside this HashMap.

User Info Hash Map

Key	Value
USERINFO_USERID	USER ID passed to Report Server for execution of report.
USERINFO_ORGID	ORGID passed to Report Server for execution of report.
USERINFO_SESSIONID	Session id.
USERINFO_SECURITY_DESCRIPTOR	Security Descriptor string.
USERINFO_CUSTOMERID	Customer id, for Service Provider deployments.
USERINFO_LOCATION	Location.
USERINFO_LOCALE	Locale setting of browser.
USERINFO_TIMESTAMP	Time of request in Java time stamp milliseconds.
USERINFO_DBNAME	String: Data base name.
USERINFO_ROLES	Roles granted to the user.

USERINFO_CONNECTION_NAME	Connection name requested to use for the report.

afterCategoryDelete

```
public void afterCategoryDelete(Object catInfo){
    System.out.println("Inside afterCategoryDelete() ");

    //Get the Category(com.intellica.client.reportutils.Category)
    //object of modified category.
    Category catObject =
          (Category)((HashMap)catInfo).get("CATEGORY_OBJECT");

    System.out.println("Category Name : "+catObject.getMenuName());
    System.out.println("Category Description :
                    "+catObject.getDescription());
    System.out.println("Category isPublic =
                    "+catObject.getIsPublic());
    System.out.println("Created Date :
                    "+catObject.getRepositDate());
    System.out.println("Category Access Rights :
                    "+catObject.getAccessRights());
    System.out.println("------- UserInfo details -------");

    //Hash Map of USER INFO which contains following values.
    HashMap userInfo =
               (HashMap)((HashMap)catInfo).get("USER INFO");
    System.out.println("USERINFO USERID ===
               "+((HashMap)userInfo).get("USERINFO USERID"));
    System.out.println("USERINFO ORGID ===
               "+((HashMap)userInfo).get("USERINFO ORGID"));
    System.out.println("USERINFO SESSIONID ===
               "+((HashMap)userInfo).get("USERINFO SESSIONID"));
    System.out.println("USERINFO SD ===
    "+((HashMap)userInfo).get("USERINFO_SECURITY_DESCRIPTOR"));
    System.out.println("USERINFO_CUSTOMERID ===
         "+((HashMap)userInfo).get("USERINFO_CUSTOMERID"));
    System.out.println("USERINFO_LOCATION ===
               "+((HashMap)userInfo).get("USERINFO_LOCATION"));
    System.out.println("USERINFO_LOCALE ===
               "+((HashMap)userInfo).get("USERINFO_LOCALE"));
    System.out.println("USERINFO_DBNAME ===
               "+((HashMap)userInfo).get("USERINFO_DBNAME"));
```

```
    System.out.println("USERINFO_ROLES ===
                "+((HashMap)userInfo).get("USERINFO_ROLES"));
    System.out.println("USERINFO_Conn_Name ===
          "+((HashMap)userInfo).get("USERINFO_CONNECTION_NAME"));
    System.out.println("End Of afterCategoryDelete()");
}
```

Place the logical code in above method.

AFTER REPORT ADD

This method will be used to Audit newly added Report Information in callback code. This event will be raised each time a new report is added.

The following method of the class will be called in this event.

void afterReportAdd(java.lang.Object reportInfo)

Event Info Hash Map

Key	Value
REPORT_LAYOUT_OBJECT	IRL (Web client's class com.impetus.intera. layout.InteraReportLayout) / ARL (Web client's com.impetus.intera.layout.adhoc. AdhocReportLayout) object of Report added.
REPORTID	Report ID.
REPORTNAME	Report Name.
CATEGORYID	Id of category in which report added.
VERSIONNO	Report Version No.
VERSIONDATE	Report Version Date.
DESIGNSTATUS	Design Status.
DEPL_TYPE	Deployment Type.
IRLVERSION	IRL version of report.
LONGDESC	Description of report.
TITLE	Report Title.
PRINTSETTINGNAME	Print Settings set for the report.
FORMAT	Default format set for the report in which report will execute.
CONNECTION_NAME	Name of connection used for report.
DSGN_MODE	Report design mode.
CONTENT_TYPE	Content Type set for the report.
REPOSITDATE	Reposit Date of an operation.
APPID	Id of a user who added report.
ORGID	Organization id of a user who added report.
ISPUBLIC	Whether report is public or not.
ISHIDDEN	Whether report is hidden or not.
REPORT_SUMMARY	Report summary.

PUBLISH_WORKFLOW_ID	Workflow id of a report.
SRC_REPORTID	Id of a parent report which will be available only for link reports.
USER_INFO	**Hash Map** of USER_INFO. Refer to USER_INFO table below for values inside this HashMap.

User Info Hash Map

Key	Value
USERINFO_USERID	USER ID passed to Report Server for execution of report.
USERINFO_ORGID	ORGID passed to Report Server for execution of report.
USERINFO_SESSIONID	Session id.
USERINFO_SECURITY_DESCRIPTOR	Security Descriptor string.
USERINFO_CUSTOMERID	Customer id, for Service Provider deployments.
USERINFO_LOCATION	Location.
USERINFO_LOCALE	Locale setting of browser.
USERINFO_TIMESTAMP	Time of request in Java time stamp milliseconds.
USERINFO_DBNAME	String: Data base name.
USERINFO_ROLES	Roles granted to the user.
USERINFO_CONNECTION_NAME	Connection name requested to use for the report.

afterReportAdd

```
public void afterReportAdd(Object repInfo){

System.out.println("********** Inside afterReportAdded() *******");
System.out.println("Report ID="+((HashMap)repInfo).get("REPORTID"));
System.out.println("Report Name="+((HashMap)repInfo).
          get("REPORTNAME"));
System.out.println("Category ID="+((HashMap)repInfo).get
                ("CATEGORYID"));
System.out.println("Version ID="+((HashMap)repInfo).get
                ("VERSIONID"));
System.out.println("Version
               No.="+((HashMap)repInfo).get("VERSIONNO"));
```

```
System.out.println("Version Date="+((HashMap)repInfo).get
                ("VERSIONDATE"));
System.out.println("Design Status="+((HashMap)repInfo).get
                ("DESIGNSTATUS"));
System.out.println("Deployment Type="+((HashMap)repInfo).get
                ("DEPL_TYPE"));
System.out.println("IRL Version="+((HashMap)repInfo).get
                ("IRLVERSION"));
System.out.println("Long Desc="+((HashMap)repInfo).get("LONGDESC"));
System.out.println("Title = "+((HashMap)repInfo).get("TITLE"));
System.out.println("Print Setting Name="+((HashMap)repInfo).get
          ("PRINTSETTINGNAME"));
System.out.println("Report Format="+((HashMap)repInfo).get
          ("FORMAT"));
System.out.println("Connection Name =
          "+((HashMap)repInfo).get("CONNECTION_NAME"));
System.out.println("Design Mode =
          "+((HashMap)repInfo).get("DSGN_MODE"));
System.out.println("Content Type =
          "+((HashMap)repInfo).get("CONTENT_TYPE"));
System.out.println("Repository Date =
          "+((HashMap)repInfo).get("REPOSITDATE"));
System.out.println("App ID = "+((HashMap)repInfo).get("APPID"));
System.out.println("Org ID = "+((HashMap)repInfo).get("ORGID"));
System.out.println("isPublic =
          "+((HashMap)repInfo).get("ISPUBLIC"));
System.out.println("isHidden =
          "+((HashMap)repInfo).get("ISHIDDEN"));
System.out.println("Report Summary =
          "+((HashMap)repInfo).get("REPORT_SUMMARY"));
System.out.println("Publish Workflow Id =
          "+((HashMap)repInfo).get("PUBLISH_WORKFLOW_ID"));
System.out.println("Source Report_ID =
          "+((HashMap)repInfo).get("SRC_REPORTID"));

System.out.println("------- UserInfo details -------");
//Hash Map of USER_INFO which contains following values.
HashMap userInfo = (HashMap)((HashMap)repInfo).get("USER_INFO");
System.out.println("USERINFO_USERID ===
    "+((HashMap)userInfo).get("USERINFO_USERID"));
System.out.println("USERINFO_ORGID ===
    "+((HashMap)userInfo).get("USERINFO_ORGID"));
System.out.println("USERINFO_SESSIONID ===
    "+((HashMap)userInfo).get("USERINFO_SESSIONID"));
```

```
System.out.println("USERINFO_SD ===
    "+((HashMap)userInfo).get("USERINFO_SECURITY_DESCRIPTOR"));
System.out.println("USERINFO_CUSTOMERID ===
    "+((HashMap)userInfo).get("USERINFO_CUSTOMERID"));
System.out.println("USERINFO_LOCATION ===
    "+((HashMap)userInfo).get("USERINFO_LOCATION"));
System.out.println("USERINFO_LOCALE ===
    "+((HashMap)userInfo).get("USERINFO_LOCALE"));
System.out.println("USERINFO_DBNAME ===
    "+((HashMap)userInfo).get("USERINFO_DBNAME"));
System.out.println("USERINFO_ROLES ===
    "+((HashMap)userInfo).get("USERINFO_ROLES"));
System.out.println("USERINFO_Conn_Name ===
    "+((HashMap)userInfo).get("USERINFO_CONNECTION_NAME"));
System.out.println("********** End Of afterReportAdded() *****");
}
```

AFTER REPORT MODIFY

This method will be used to Audit modified Report Information in callback code. This event will be raised each time any existing report is modified.

The following method of the class will be called in this event.

void afterReportModify(java.lang.Object reportInfo)

Event Info Hash Map

Key	Value
REPORT_LAYOUT_OBJECT	IRL (Web client's class com.impetus.intera. layout.InteraReportLayout) / ARL (Web client's com.impetus.intera.layout.adhoc. AdhocReportLayout) object of Report modified. If Report Design is updated, then only this attribute will contain IRL/ARL object otherwise it will contain null.
REPORTID	Report ID.
REPORTNAME	Report Name.
CATEGORYID	Id of category in which report modifed.
VERSIONNO	Report Version No.
VERSIONDATE	Report Version Date.
DESIGNSTATUS	Design Status.
DEPL_TYPE	Deployment Type.
IRLVERSION	IRL version of report.
LONGDESC	Description of report.
TITLE	Report Title.
PRINTSETTINGNAME	Print Settings set for the report.

FORMAT	Default format set for the report in which report will execute.
CONNECTION_NAME	Name of connection used for report.
DSGN_MODE	Report design mode.
CONTENT_TYPE	Content Type.
REPOSITDATE	Reposit Date of an operation.
APPID	Id of a user who added report.
ORGID	Organization id of a user who added report.
ISPUBLIC	Whether report is public or not.
ISHIDDEN	Whether report is hidden or not.
REPORT_SUMMARY	Report summary.
PUBLISH_WORKFLOW_ID	Workflow id of a report.
SRC_REPORTID	Id of a parent or link report.
USER_INFO	**Hash Map** of USER_INFO. Refer to USER_INFO table below for values inside this HashMap.

User Info Hash Map

Key	Value
USERINFO_USERID	USER ID passed to Report Server for execution of report.
USERINFO_ORGID	ORGID passed to Report Server for execution of report.
USERINFO_SESSIONID	Session id.
USERINFO_SECURITY_DESCRIPTOR	Security Descriptor string.
USERINFO_CUSTOMERID	Customer id, for Service Provider deployments.
USERINFO_LOCATION	Location.
USERINFO_LOCALE	Locale setting of browser.
USERINFO_TIMESTAMP	Time of request in Java time stamp milliseconds.
USERINFO_DBNAME	String: Data base name.
USERINFO_ROLES	Roles granted to the user.
USERINFO_CONNECTION_NAME	Connection name requested to use for the report.

afterReportModify

```
public void afterReportModify (Object repInfo){
```

```
System.out.println("********** Inside afterReportModified() ****");
System.out.println("Report ID="+((HashMap)repInfo).get("REPORTID"));
System.out.println("Report Name="+((HashMap)repInfo).
               get("REPORTNAME"));
System.out.println("Category ID="+((HashMap)repInfo).get
               ("CATEGORYID"));
System.out.println("Version ID="+((HashMap)repInfo).get
               ("VERSIONID"));
System.out.println("Version
               No.="+((HashMap)repInfo).get("VERSIONNO"));
System.out.println("Version Date="+((HashMap)repInfo).get
               ("VERSIONDATE"));
System.out.println("Design Status="+((HashMap)repInfo).get
               ("DESIGNSTATUS"));
System.out.println("Deployment Type="+((HashMap)repInfo).get
               ("DEPL_TYPE"));
System.out.println("IRL Version="+((HashMap)repInfo).get
               ("IRLVERSION"));
System.out.println("Long Desc="+((HashMap)repInfo).get("LONGDESC"));
System.out.println("Title = "+((HashMap)repInfo).get("TITLE"));
System.out.println("Print Setting Name="+((HashMap)repInfo).get
          ("PRINTSETTINGNAME"));
System.out.println("Report Format="+((HashMap)repInfo).get
          ("FORMAT"));
System.out.println("Connection Name =
          "+((HashMap)repInfo).get("CONNECTION_NAME"));
System.out.println("Design Mode =
          "+((HashMap)repInfo).get("DSGN_MODE"));
System.out.println("Content Type =
          "+((HashMap)repInfo).get("CONTENT_TYPE"));
System.out.println("Repository Date =
          "+((HashMap)repInfo).get("REPOSITDATE"));
System.out.println("App ID = "+((HashMap)repInfo).get("APPID"));
System.out.println("Org ID = "+((HashMap)repInfo).get("ORGID"));
System.out.println("isPublic =
          "+((HashMap)repInfo).get("ISPUBLIC"));
System.out.println("isHidden =
          "+((HashMap)repInfo).get("ISHIDDEN"));
System.out.println("Report Summary =
          "+((HashMap)repInfo).get("REPORT_SUMMARY"));
System.out.println("Publish Workflow Id =
          "+((HashMap)repInfo).get("PUBLISH_WORKFLOW_ID"));
System.out.println("Source Report_ID =
          "+((HashMap)repInfo).get("SRC_REPORTID"));
```

```
System.out.println("------- UserInfo details -------");
//Hash Map of USER_INFO which contains following values.
HashMap userInfo = (HashMap)((HashMap)repInfo).get("USER_INFO");
System.out.println("USERINFO_USERID ===
    "+((HashMap)userInfo).get("USERINFO_USERID"));
System.out.println("USERINFO_ORGID ===
    "+((HashMap)userInfo).get("USERINFO_ORGID"));
System.out.println("USERINFO_SESSIONID ===
    "+((HashMap)userInfo).get("USERINFO_SESSIONID"));
System.out.println("USERINFO_SD ===
    "+((HashMap)userInfo).get("USERINFO_SECURITY_DESCRIPTOR"));
System.out.println("USERINFO_CUSTOMERID ===
    "+((HashMap)userInfo).get("USERINFO_CUSTOMERID"));
System.out.println("USERINFO_LOCATION ===
    "+((HashMap)userInfo).get("USERINFO_LOCATION"));
System.out.println("USERINFO_LOCALE ===
    "+((HashMap)userInfo).get("USERINFO_LOCALE"));
System.out.println("USERINFO_DBNAME ===
    "+((HashMap)userInfo).get("USERINFO_DBNAME"));
System.out.println("USERINFO_ROLES ===
    "+((HashMap)userInfo).get("USERINFO_ROLES"));
System.out.println("USERINFO_Conn_Name ===
    "+((HashMap)userInfo).get("USERINFO_CONNECTION_NAME"));
System.out.println("********** End Of afterReportModified() *****");
}
```

Place the logical code in above method.

AFTER REPORT DELETE

This method will be used to Audit deleted Report Information in callback code. This event will be raised each time any existing report is deleted.

The following method of the class will be called in this event.

void afterReportDelete(java.lang.Object reportInfo)

Event Info Hash Map

Key	Value
REPORTID	Report ID.
REPORTNAME	Report Name.
CATEGORYID	Id of category from which report deleted.
VERSIONNO	Report Version No.
VERSIONDATE	Report Version Date.
DESIGNSTATUS	Design Status.
DEPL_TYPE	Deployment Type.

IRLVERSION	IRL version of report.
LONGDESC	Description of report.
TITLE	Report Title.
PRINTSETTINGNAME	Print Settings set for the report.
FORMAT	Default format set for the report in which report will execute.
CONNECTION_NAME	Name of connection used for report.
DSGN_MODE	Report design mode.
CONTENT_TYPE	Content Type.
REPOSITDATE	Reposit Date of an operation.
APPID	Id of a user who deleted report.
ORGID	Organization id of a user who deleted report.
ISPUBLIC	Whether report is public or not.
ISHIDDEN	Whether report is hidden or not.
REPORT_SUMMARY	Report summary.
PUBLISH_WORKFLOW_ID	Workflow id of a report.
SRC_REPORTID	Id of a parent or link report.
USER_INFO	**Hash Map** of USER_INFO. Refer to USER_INFO table below for values inside this HashMap.

User Info Hash Map

Key	**Value**
USERINFO_USERID	USER ID passed to Report Server for execution of report.
USERINFO_ORGID	ORGID passed to Report Server for execution of report.
USERINFO_SESSIONID	Session id.
USERINFO_SECURITY_DESCRIPTOR	Security Descriptor string.
USERINFO_CUSTOMERID	Customer id, for Service Provider deployments.
USERINFO_LOCATION	Location.
USERINFO_LOCALE	Locale setting of browser.
USERINFO_TIMESTAMP	Time of request in Java time stamp milliseconds.
USERINFO_DBNAME	String: Data base name.
USERINFO_ROLES	Roles granted to the user.
USERINFO_CONNECTION_NAME	Connection name requested to use for the report.

afterReportDelete

```
public void afterReportDelete(Object repInfo){

System.out.println("**********      Inside      afterReportDeleted()
*******");
System.out.println("Report ID="+((HashMap)repInfo).get("REPORTID"));
System.out.println("Report Name="+((HashMap)repInfo).
          get("REPORTNAME"));
System.out.println("Category ID="+((HashMap)repInfo).get
                ("CATEGORYID"));
System.out.println("Version ID="+((HashMap)repInfo).get
                ("VERSIONID"));
System.out.println("Version
               No.="+((HashMap)repInfo).get("VERSIONNO"));
System.out.println("Version Date="+((HashMap)repInfo).get
                ("VERSIONDATE"));
System.out.println("Design Status="+((HashMap)repInfo).get
                ("DESIGNSTATUS"));
System.out.println("Deployment Type="+((HashMap)repInfo).get
                ("DEPL_TYPE"));
System.out.println("IRL Version="+((HashMap)repInfo).get
                ("IRLVERSION"));
System.out.println("Long Desc="+((HashMap)repInfo).get("LONGDESC"));
System.out.println("Title = "+((HashMap)repInfo).get("TITLE"));
System.out.println("Print Setting Name="+((HashMap)repInfo).get
          ("PRINTSETTINGNAME"));
System.out.println("Report Format="+((HashMap)repInfo).get
          ("FORMAT"));
System.out.println("Connection Name =
          "+((HashMap)repInfo).get("CONNECTION_NAME"));
System.out.println("Design Mode =
          "+((HashMap)repInfo).get("DSGN_MODE"));
System.out.println("Content Type =
          "+((HashMap)repInfo).get("CONTENT_TYPE"));
System.out.println("Repository Date =
          "+((HashMap)repInfo).get("REPOSITDATE"));
System.out.println("App ID = "+((HashMap)repInfo).get("APPID"));
System.out.println("Org ID = "+((HashMap)repInfo).get("ORGID"));
System.out.println("isPublic =
          "+((HashMap)repInfo).get("ISPUBLIC"));
System.out.println("isHidden =
          "+((HashMap)repInfo).get("ISHIDDEN"));
```

```
System.out.println("Report Summary =
          "+((HashMap)repInfo).get("REPORT_SUMMARY"));
System.out.println("Publish Workflow Id =
          "+((HashMap)repInfo).get("PUBLISH_WORKFLOW_ID"));
System.out.println("Source Report_ID =
          "+((HashMap)repInfo).get("SRC_REPORTID"));

System.out.println("------- UserInfo details -------");
//Hash Map of USER_INFO which contains following values.
HashMap userInfo = (HashMap)((HashMap)repInfo).get("USER_INFO");
System.out.println("USERINFO_USERID ===
    "+((HashMap)userInfo).get("USERINFO_USERID"));
System.out.println("USERINFO_ORGID ===
    "+((HashMap)userInfo).get("USERINFO_ORGID"));
System.out.println("USERINFO_SESSIONID ===
    "+((HashMap)userInfo).get("USERINFO_SESSIONID"));
System.out.println("USERINFO_SD ===
    "+((HashMap)userInfo).get("USERINFO_SECURITY_DESCRIPTOR"));
System.out.println("USERINFO_CUSTOMERID ===
    "+((HashMap)userInfo).get("USERINFO_CUSTOMERID"));
System.out.println("USERINFO_LOCATION ===
    "+((HashMap)userInfo).get("USERINFO_LOCATION"));
System.out.println("USERINFO_LOCALE ===
    "+((HashMap)userInfo).get("USERINFO_LOCALE"));
System.out.println("USERINFO_DBNAME ===
    "+((HashMap)userInfo).get("USERINFO_DBNAME"));
System.out.println("USERINFO_ROLES ===
    "+((HashMap)userInfo).get("USERINFO_ROLES"));
System.out.println("USERINFO_Conn_Name ===
    "+((HashMap)userInfo).get("USERINFO_CONNECTION_NAME"));
System.out.println("********** End Of afterReportDeleted() *****");
}
```

Place the logical code in above method.

ROMGMTEVENTS

Intellicus facilitates Auditing callback events mechanism for Report Object Management Operations by "calling your code" system. Intellicus provides a class "ReportObjectMgmtEvents"

Configuration

For implementing **ROMgmtEvents**, **EVENTSHANDLER TYPE** should be: ROMGMTEVENTS.

Following code shows the XML entry having configuration of UMM Events.

```
<EVENTSHANDLER TYPE=" ROMGMTEVENTS">
  <CALLBACK CALLTYPE="1">
        <IMPLEMENTER TYPE="1">
              <ATTRS TYPE="1">
                    <ATTR NAME="PATH">
                          <VALUE>com.mypackage.myclass</VALUE>
                    </ATTR>
              </ATTRS>
        </IMPLEMENTER>
  </CALLBACK>
</ EVENTSHANDLER>
```

During callback events process Intellicus Report Server will call these methods:

1. void afterReportObjectAdd(Object reportObjectInfo)**:** Provide the callback event Infomation after a new Report Object(Query Object/Parameter Object) is added in Intellicus. Provide this information to the java class using HashMap as object.

2. void afterReportObjectModify(Object reportObjectInfo)**:** Provide the callback event Infomation after any existing Report Object(Query Object/Parameter Object) is modified in Intellicus. Provide this information to the java class using HashMap as object.

3. void afterReportObjectDelete(Object reportObjectInfo)**:** Provide the callback event Infomation after any Report Object(Query Object/Parameter Object) is deleted in Intellicus. Provide this information to the java class using HashMap as object.

Sample Implementation code

```
import com.impetus.interaj.callback.ROMgmtEvents;

public class ReportMgmtCallbackImpl extends ROMgmtEvents {

}
```

AFTER REPORTOBJECT ADD

This method will be used to Audit newly added Report Object (Query Object/Parameter Object) Information in callback code. This event will be raised each time a new report object is added.

The following method of the class will be called in this event.

void afterReportObjectAdd (java.lang.Object reportObjectInfo**)**

Event Info Hash Map

Key	Value
REPORT_OBJECT	Report Object (Web client's class com.impetus.intera.reportobjects. ReportObject) of QO/PO added.
USER_INFO	**Hash Map** of USER_INFO. Refer to USER_INFO table below for values inside this HashMap.

User Info Hash Map

Key	Value
USERINFO_USERID	USER ID passed to Report Server for execution of report.
USERINFO_ORGID	ORGID passed to Report Server for execution of report.
USERINFO_SESSIONID	Session id.
USERINFO_SECURITY_DESCRIPTOR	Security Descriptor string.
USERINFO_CUSTOMERID	Customer id, for Service Provider deployments.
USERINFO_LOCATION	Location.
USERINFO_LOCALE	Locale setting of browser.
USERINFO_TIMESTAMP	Time of request in Java time stamp milliseconds.
USERINFO_DBNAME	String: Data base name.
USERINFO_ROLES	Roles granted to the user.
USERINFO_CONNECTION_NAME	Connection name requested to use for the report.

afterReportObjectAdd

```
public void afterReportObjectAdd(Object ROInfo){

System.out.println("********** Inside afterReportObjectAdd() ***");

//Get the object of ReportObject that is newly added.
ReportObject reportObject =
(ReportObject)((HashMap)ROInfo).get("REPORT_OBJECT");

System.out.println("Report Object ID : "+reportObject.getId());
System.out.println("Report Object Name : "+reportObject.getName());
```

```
System.out.println("Report Object Type = "+reportObject.getType());
System.out.println("Report Object Description :
    "+reportObject.getDescription());

System.out.println("------- UserInfo details -------");
//Hash Map of USER_INFO which contains following values.
HashMap userInfo = (HashMap)((HashMap)ROInfo).get("USER_INFO");
System.out.println("USERINFO_USERID ===
    "+((HashMap)userInfo).get("USERINFO_USERID"));
System.out.println("USERINFO_ORGID ===
    "+((HashMap)userInfo).get("USERINFO_ORGID"));
System.out.println("USERINFO_SESSIONID ===
    "+((HashMap)userInfo).get("USERINFO_SESSIONID"));
System.out.println("USERINFO_SD ===
    "+((HashMap)userInfo).get("USERINFO_SECURITY_DESCRIPTOR"));
System.out.println("USERINFO_CUSTOMERID ===
    "+((HashMap)userInfo).get("USERINFO_CUSTOMERID"));
System.out.println("USERINFO_LOCATION ===
    "+((HashMap)userInfo).get("USERINFO_LOCATION"));
System.out.println("USERINFO_LOCALE ===
    "+((HashMap)userInfo).get("USERINFO_LOCALE"));
System.out.println("USERINFO_DBNAME ===
    "+((HashMap)userInfo).get("USERINFO_DBNAME"));
System.out.println("USERINFO_ROLES ===
    "+((HashMap)userInfo).get("USERINFO_ROLES"));
System.out.println("USERINFO_Conn_Name ===
    "+((HashMap)userInfo).get("USERINFO_CONNECTION_NAME"));

System.out.println("********** End Of afterReportObjectAdd() ***");
}
```

Place the logical code in above method.

AFTER REPORTOBJECTMODIFY

This method will be used to Audit modified Report Object (Query Object/Parameter Object) Information in callback code. This event will be raised each time any existing report object is modified.

The following method of the class will be called in this event.

void afterReportObjectModify (java.lang.Object reportInfo)

Event Info Hash Map

Key	Value
REPORT_OBJECT	Report Object (Web client's class

	com.impetus.intera.reportobjects. ReportObject) of QO/PO modified.
USER_INFO	**Hash Map** of USER_INFO. Refer to USER_INFO table below for values inside this HashMap.

User Info Hash Map

Key	Value
USERINFO_USERID	USER ID passed to Report Server for execution of report.
USERINFO_ORGID	ORGID passed to Report Server for execution of report.
USERINFO_SESSIONID	Session id.
USERINFO_SECURITY_DESCRIPTOR	Security Descriptor string.
USERINFO_CUSTOMERID	Customer id, for Service Provider deployments.
USERINFO_LOCATION	Location.
USERINFO_LOCALE	Locale setting of browser.
USERINFO_TIMESTAMP	Time of request in Java time stamp milliseconds.
USERINFO_DBNAME	String: Data base name.
USERINFO_ROLES	Roles granted to the user.
USERINFO_CONNECTION_NAME	Connection name requested to use for the report.

afterReportObjectModify

```
public void afterReportObjectModify(Object ROInfo){

System.out.println("***** Inside afterReportObjectModify() *");

//Get the object of ReportObject that is modified.
ReportObject reportObject =
    (ReportObject)((HashMap)ROInfo).get("REPORT_OBJECT");

System.out.println("Report Object ID : "+reportObject.getId());
System.out.println("Report Object Name : "+reportObject.getName());
System.out.println("Report Object Type = "+reportObject.getType());
System.out.println("Report Object Description :
    "+reportObject.getDescription());
```

```
System.out.println("------- UserInfo details -------");
//Hash Map of USER_INFO which contains following values.
HashMap userInfo = (HashMap)((HashMap)ROInfo).get("USER_INFO");
System.out.println("USERINFO_USERID ===
    "+((HashMap)userInfo).get("USERINFO_USERID"));
System.out.println("USERINFO_ORGID ===
    "+((HashMap)userInfo).get("USERINFO_ORGID"));
System.out.println("USERINFO_SESSIONID ===
    "+((HashMap)userInfo).get("USERINFO_SESSIONID"));
System.out.println("USERINFO_SD ===
    "+((HashMap)userInfo).get("USERINFO_SECURITY_DESCRIPTOR"));
System.out.println("USERINFO_CUSTOMERID ===
    "+((HashMap)userInfo).get("USERINFO_CUSTOMERID"));
System.out.println("USERINFO_LOCATION ===
    "+((HashMap)userInfo).get("USERINFO_LOCATION"));
System.out.println("USERINFO_LOCALE ===
    "+((HashMap)userInfo).get("USERINFO_LOCALE"));
System.out.println("USERINFO_DBNAME ===
    "+((HashMap)userInfo).get("USERINFO_DBNAME"));
System.out.println("USERINFO_ROLES ===
    "+((HashMap)userInfo).get("USERINFO_ROLES"));
System.out.println("USERINFO_Conn_Name ===
    "+((HashMap)userInfo).get("USERINFO_CONNECTION_NAME"));

System.out.println("***** End Of afterReportObjectModify() ****");
}
```

Place the logical code in above method.

AFTER REPORTOBJECT DELETE

This method will be used to Audit deleted Report Object(Query Object/Parameter Object) Information in callback code. This event will be raised each time any report object is deleted.

The following method of the class will be called in this event.

void afterReportObjectDelete (java.lang.Object reportInfo)

Event Info Hash Map

Key	Value
REPORT_OBJECT	Report Object (Web client's class com.impetus.intera.reportobjects. ReportObject) of QO/PO deleted.
USER_INFO	**Hash Map** of USER_INFO.

	Refer to USER_INFO table below for values inside this HashMap.

User Info Hash Map

Key	Value
USERINFO_USERID	USER ID passed to Report Server for execution of report.
USERINFO_ORGID	ORGID passed to Report Server for execution of report.
USERINFO_SESSIONID	Session id.
USERINFO_SECURITY_DESCRIPTOR	Security Descriptor string.
USERINFO_CUSTOMERID	Customer id, for Service Provider deployments.
USERINFO_LOCATION	Location.
USERINFO_LOCALE	Locale setting of browser.
USERINFO_TIMESTAMP	Time of request in Java time stamp milliseconds.
USERINFO_DBNAME	String: Data base name.
USERINFO_ROLES	Roles granted to the user.
USERINFO_CONNECTION_NAME	Connection name requested to use for the report.

afterReportObjectDelete

```
public void afterReportObjectDelete(Object ROInfo){

System.out.println("******* Inside afterReportObjectDelete() ****");

//Get the object of ReportObject that is Deleted.
ReportObject reportObject =
    (ReportObject)((HashMap)ROInfo).get("REPORT_OBJECT");

System.out.println("Report Object ID : "+reportObject.getId());
System.out.println("Report Object Name : "+reportObject.getName());
System.out.println("Report Object Type = "+reportObject.getType());
System.out.println("Report Object Description :
    "+reportObject.getDescription());

System.out.println("------- UserInfo details -------");
//Hash Map of USER_INFO which contains following values.
HashMap userInfo = (HashMap)((HashMap)ROInfo).get("USER_INFO");
```

```
System.out.println("USERINFO_USERID ===
    "+((HashMap)userInfo).get("USERINFO_USERID"));
System.out.println("USERINFO_ORGID ===
    "+((HashMap)userInfo).get("USERINFO_ORGID"));
System.out.println("USERINFO_SESSIONID ===
    "+((HashMap)userInfo).get("USERINFO_SESSIONID"));
System.out.println("USERINFO_SD ===
    "+((HashMap)userInfo).get("USERINFO_SECURITY_DESCRIPTOR"));
System.out.println("USERINFO_CUSTOMERID ===
    "+((HashMap)userInfo).get("USERINFO_CUSTOMERID"));
System.out.println("USERINFO_LOCATION ===
    "+((HashMap)userInfo).get("USERINFO_LOCATION"));
System.out.println("USERINFO_LOCALE ===
    "+((HashMap)userInfo).get("USERINFO_LOCALE"));
System.out.println("USERINFO_DBNAME ===
    "+((HashMap)userInfo).get("USERINFO_DBNAME"));
System.out.println("USERINFO_ROLES ===
    "+((HashMap)userInfo).get("USERINFO_ROLES"));
System.out.println("USERINFO_Conn_Name ===
    "+((HashMap)userInfo).get("USERINFO_CONNECTION_NAME"));

System.out.println("****** End Of afterReportObjectDelete()******");
}
```

Place the logical code in above method.

DynamicQOEvents

DynamicQOEvents is to dynamically modify the existing QO or get an external QO.
This event can be implemented to Add/Delete columns and/or modify column attributes of the associated Query object, before the Report Execution.
Also, it enables fetching of external Query Object for running a Report i.e. User can create QO dynamically without saving it in Intellicus Repository.

Configuration

For implementing **DynamicQOEvents**, **EVENTSHANDLER TYPE** should be: DYNAMICQOEVENTS.

Following code shows the XML entry having configuration of UMM Events.

```
<EVENTSHANDLER TYPE="DYNAMICQOEVENTS">
       <!-- The callback type attribute defines the call back
mode implemented
       Supported mode is 1 which is LOCAL callback mode  -->
            <CALLBACK CALLTYPE="1">
```

```
            <!-- The callback implementer attribute defines the
call back implementor type
            Supported mode is 1 which is JAVA callback
implemeter type    -->
            <IMPLEMENTER TYPE="1">
                  <ATTRS TYPE="1">
                        <ATTR NAME="PATH">
                        <!--Specify the name of the class that is
extending the base class. Make sure that the class is present in
the classpath of the Report Server.-->
                              <VALUE>DynamicQOCallbackImpl</VALUE>
                        </ATTR>
                  </ATTRS>
          </IMPLEMENTER>
       </CALLBACK>
</EVENTSHANDLER>
```

During callback events process Intellicus Report Server will call these methods:

1. void **getDynamicColumns**(java.util.Map<java.lang.String,java.lang.Object> queryObjectInfo)**:** This method will be used to get columns for dynamic columns.
2. void **getExternalQueryObject** (java.util.Map<java.lang.String,java.lang.Object> queryObjectInfo)**:** This method will be used to fetch external query object.

Sample Implementation code

```
import com.impetus.interaj.callback.DynamicQOEvents;

public class DynamicQOCallbackImpl extends DynamicQOEvents
{
}
```

Get Dynamic Columns

This method will be used to get columns for dynamic columns. The method will be called when dynamic columns are required

The following method of the class will be called in this event.

public void **getDynamicColumns**
(java.util.**Map**<java.lang.String,java.lang.Object> queryObjectInfo)
throws java.lang.Exception

Parameter-

queryObjectInfo Map- This is generally a HashMap that contains Report Object Information. The HashMap contains following values:

queryObjectInfo Map

Key	Value
QUERY_OBJECT	com.impetus.intera.reportobjects.QueryObject
CONNECTION	java.sql.connection class object => The connection used to execute the report.
USER_INFO HashMap	Hash Map of USER_INFO which contains following values.
SYS_PARAMS	Hash Map of all System Parameters passed to the Report. KEY of each element is the name of system parameter and VALUE is the string of parameter value
USER_PARAMS	Hash Map of all user parameters passed to the report. KEY of each element is the parameter name and VALUE is the string of parameter value.

USER_INFO HashMap

Key	Value
USERINFO_USERID	USER ID passed to Report Server for execution of report
USERINFO_ORGID	ORGID passed to Report Server for execution of report.
USERINFO_SESSIONID	Session id.
USERINFO_SECURITY_DESCRIPTOR	Security Descriptor string.
USERINFO_CUSTOMERID	Customer id, for Service Provider deployments.
USERINFO_LOCATION	Location.
USERINFO_LOCALE	Locale setting of browser.
USERINFO_TIMESTAMP	Time of request in Java time stamp milliseconds.
USERINFO_DBNAME	String: Data base name.
USERINFO_ROLES	Roles granted to the user.
USERINFO_CONNECTION_NAME	Connection name requested to use for the report.

getDynamicColumns

```
public void getDynamicColumns(Map<String,Object> queryObjectInfo)
throws Exception
{
          System.out.println("Inside getDynamicColumns");

        //Get query object
          QueryObject queryObject = (QueryObject)
queryObjectInfo.get("QUERY_OBJECT");

        // Suppose a query object contains below columns
        // CUSTOMER.CUST_NO, CUSTOMER.CUSTOMER, CUSTOMER.STATUS,
        // CUSTOMER.COMMENTS, CUSTOMER.TYPE
        // Then
        // Remove CUSTOMER.COMMENTS & CUSTOMER.TYPE dynamically by
call back
        queryObject.removeColumn("TYPE");
        System.out.println("removed Column : TYPE");

        queryObject.removeColumn("COMMENTS");
        System.out.println("removed Column : COMMENTS");

        System.out.println("End of getDynamicColumns");
}
```

Place the logical code in above method.

Get External Query Object

This method will be used to fetch external query object. The following method of the class will be called in this event.

public void
getExternalQueryObject(java.util.**Map**<java.lang.String,java.lang.Object> queryObjectInfo)
throws java.lang.Exception

Parameter-
queryObjectInfo Map- This is generally a HashMap that contains Report Object Information. The HashMap contains following values:

queryObjectInfo Map

Key	Value

Key	Value
QUERY_ID: String	Query Object ID
QUERY_NAME: String	Query Object Name
QUERY_OBJECT	com.impetus.intera.reportobjects.QueryObject
CONNECTION: java.sql.connection	class object => The connection used to execute the report.
CONNECTION_NAME: String	The Intellicus connection name used in query object.
USER_INFO HashMap	Hash Map of USER_INFO which contains following values.
SYS_PARAMS	Hash Map of all System Parameters passed to the Report. KEY of each element is the name of system parameter and VALUE is the string of parameter value
USER_PARAMS	Hash Map of all user parameters passed to the report. KEY of each element is the parameter name and VALUE is the string of parameter value.

USER_INFO HashMap

Key	Value
USERINFO_USERID	USER ID passed to Report Server for execution of report
USERINFO_ORGID	ORGID passed to Report Server for execution of report.
USERINFO_SESSIONID	Session id.
USERINFO_SECURITY_DESCRIPTOR	Security Descriptor string.
USERINFO_CUSTOMERID	Customer id, for Service Provider deployments.
USERINFO_LOCATION	Location.
USERINFO_LOCALE	Locale setting of browser.
USERINFO_TIMESTAMP	Time of request in Java time stamp milliseconds.
USERINFO_DBNAME	String: Data base name.
USERINFO_ROLES	Roles granted to the user.
USERINFO_CONNECTION_NAME	Connection name requested to use for the report.

getExternalQueryObject

```
public void getExternalQueryObject(Map<String,Object>
queryObjectInfo) throws Exception
{
          System.out.println("Inside getExternalQueryObject");

        //set the query object ID & name
          String queryObjectID = "QueryObjectDel";
          String queryObjectName = "QueryObjectDel";

          //create SQL query
        String sqlQuery = "select COUNTRY.* from COUNTRY";

        System.out.println("Query Id = "+queryObjectID);
        System.out.println("Query Name = "+queryObjectName);
        System.out.println("SQL Query = "+sqlQuery);

        //get the connection to execute query
        Connection connection =
(Connection)queryObjectInfo.get("CONNECTION");
        String connectionName =
(String)queryObjectInfo.get("CONNECTION_NAME");

        System.out.println("connectionName = "+connectionName);

        if(connectionName==null)
        {
          connectionName="";
        }
        IResultSet iResultSet = this.executeQuery(sqlQuery,
connection);

        //Create Query Object
        QueryObject qo =
QueryObject.createQueryObject(queryObjectName,
                Transformation.FetchStep.Source.SQL, sqlQuery, null,
                connectionName, null, iResultSet);

        qo.setId(queryObjectID);

        //set the qo in info map
        queryObjectInfo.put("QUERY_OBJECT", qo);
```

```
        System.out.println("End of getExternalQueryObject");
}
```

Place the logical code in above method.

Note: Please refer sample code available at <Intellicus_Install_Path>\SampleCodes\CallBack APIs\CallBack Events

JVista API

The document describes various properties, methods and events exposed by the JVista Report Viewer. It also provides sample code snippets for a smooth understanding of the API. The API is divided into two portions:

- Standard: Provides API calls for quick and loose integration.
- Advanced: Provides API calls for advanced viewer configuration. This includes changing the look and feel and the viewer behavior etc.

Standard API

This section describes the standard API provided by the JVista viewer. All the API functions are methods of the JVista class. The user must instantiate the object of JVista class for using the JVista API. The JVista class extends the JSplitPane swing container. This helps the user to integrate with their existing swing applications.

Methods

SetReportProperties

Syntax

```
SetReportProperties(String reportID, String categoryID)
          throws JVistaException
```

Used to provide details of the report.

Params

- **ReportID:** The ID of the report to be executed. In case when reports are stored in the database it is the primary key of the row containing the report template file (table name: intera_rlayout). When reports are stored in file system then it is the relative path of the file name with respect to the reportlayout folder set in the Intellica Report Engine. This path also includes the category folder names.
- **CategoryID:** The categoryID of the report. In case when reports are stored in the database it is the primary key of the row containing the category of the reportID parameter (table name: intera_rcategory). When reports are stored in files then it is the relative path of the category folder with respect to the reportlayout folder set in the Intellica Report Engine.

SetTempDirectory

Syntax

```
(String tmpPathName) throws JVistaException
```

Used to set the temp folder path. The JVista viewer uses this path to create temporary files.

Params

- **TmpPathName:** The name of the temporary directory.

SetURL

Syntax

```
SetURL(URLConnection urlConn)
```

Used to set the URL from which the viewer will read the report.

Params

- **UrlConn:** Object of type java.net.URLConnection

setConfigProperty

Syntax

```
setConfigProperty (int key, Object value)
```

Used to set the Configuration related properties.

Params

- **key:** Object of type java.net.URLConnection
- **value:** Value of the key type Object.

Example: setConfigProperty(JVistaEnums.JVistaViewer.BACKCOLOR,Color.gray);

setPageChunkSize

```
setConfigProperty(JVistaEnums.JVistaViewer.BACKCOLOR,Color.gray);
```

Syntax

```
setPageChunkSize(int size)
```

Used to set the Page Chunk Size.

Params

- **size:** Size of Page Chunk

Example: setPageChunkSize(10)

setHostName

Syntax

```
setHostName(String hostName)
```

Used to set the IP Address of the Report Engine.

Params

- **hostName:** IP-Address of the Report Engine.

setHostPort

Syntax

```
setHostPort(int hostPort)
```

Used to set the Report Engine port

Params

- **hostPort:** Port Number of the Report Engine.

setReport

Syntax

```
setReport(String reportID, String categoryID, String menuName)
```

Used to set the Report Details that is to be viewed in JVista.

Params

- **reportID:** Report ID of the Report that is to be viewed in JVista format
- **categoryID:** Category Id of the Category to which the Report belongs.
- **menuName:** Name of the Report.

SetURL

Overloaded function to take URL name as a parameter instead of an object. Used to set the URL from which the viewer will read the report.

Syntax

```
SetURL(String urlName)
```

Params

- **urlName:** The fully qualified URL string.

Initialize

Initializes the JVista object.

Syntax

```
Initialize (String sessionID, JVistaEventInterface eInterface)
           throws JVistaException
```

Params

- **sessionID:** Used to uniquely identify the session for communication with the URL set the setURL methods.
- **eInterface:** The callback interface used for providing report events to the host application. The events are described in the Events section below.

ShowReport

Displays report on the JVista viewer.

Syntax

```
ShowReport(String reportInstanceID, boolean showParamForm)
          throws JVistaException
```

Params

ReportInstanceID: This is for future use. Must be passed as blank string (“”). ShowParamForm: Determines whether to display the report parameter form or not. Values (true/false).

Events

JVista can notify the application using it of certain events that take place during viewing a report. For this, the application has to register a class-implementing interface JvistaEventInterface with the JVista instance in use. Following are the methods in the interface:

public void onReportLoadComplete()

Notifies the observer of the completion of report download from the server.

public void onReportPrintComplete()

Notifies the observer of the completion of a print job fired from the viewer.

public void onReportPageChanged(int previousPage , int currentPage)

Notifies an observer whenever the page being viewed by the user is changed. The first argument is the number of the old page and the second one is the number of the new page.

public void onError (JVistaException e)

Notifies an observer whenever there is an error during processing. The argument is the JVistaException object. It contains methods to retrieve the error code and error string.

public void viewerInitialized(void)

Notifies an observer whenever the viewer is loaded. This event must used to set the size of the viewer. See code sample below.

Code Sample

Given below is a completely functional sample code to load and use the JVista viewer.

```
import com.intellica.jvista.JVista;
import com.intellica.jvista.ui.core.JVistaEventInterface;
import com.intellica.jvista.common.JVistaException;

import javax.swing.JFrame;
import javax.swing.UIManager;

import java.net.URL;
import java.net.URLConnection;

public class JVistaTest implements JVistaEventInterface
{

    JFrame frame;
    JVista jvista;

    public JVistaTest()
    {
    }

    public void initialize() throws JVistaException
    {
        //create a JFrame and initialize it
        this.frame = new JFrame();
        frame.pack();
        frame.setVisible(true);

        //Create a URL connection object
        String                    URLString                    =
"http://localhost/celer/IntellicaController";
        URLConnection               urlConnection              =
getURLConnection(URLString);

        //create a JVista object and initialize it
        this.jvista = new JVista();
        this.jvista.setTempDirectory("c:\\temp");

        this.jvista.initialize(this);
```

```
this.jvista.setReportPropeties("jvista/sample1","jvista");

//this.jvista.setURL("http://localhost/celer/IntellicaController"
);
        this.jvista.setURL(urlConnection);
    }

    private URLConnection getURLConnection(String URLString)
    {
        URLConnection urlConnection = null;
        try
        {
            URL tmpURL = new URL(URLString);
            urlConnection = tmpURL.openConnection();
            return urlConnection;
        }
        catch(Exception e)
        {
            System.out.println ("cannot create URL object");
            System.exit(1);
        }
        return urlConnection;
    }

    public void show() throws JVistaException
    {
        this.jvista.showReport("",true);
    }

    public void reportLoadComplete()
    {
        System.out.println("In reportLoad Complete");
    }
    public void reportPrintComplete()
    {
        System.out.println("In report Print  ");
    }

     public     void     reportPageChanged(int     previousPage,int
currentPage)
     {
        System.out.println("In reportPage Changed");
     }
```

```
    public void onError(JVistaException exception)
    {
        System.out.println("In error ");
    }

    public void viewerInitialized()
    {
        System.out.println("In viewer loaded");

        this.frame.getContentPane().add(this.jvista);

this.frame.getContentPane().setSize(this.jvista.getPreferredSize(
));
        this.frame.setSize(this.jvista.getPreferredSize());
    }

    public static void main(String args[]) throws Exception
    {

        //set the look and feel

UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.Windows
LookAndFeel");



        //create the JVistaTest object
        JVistaTest jVistaTest = new JVistaTest();
        jVistaTest.initialize();

        //show the report
        jVistaTest.show();
}
```

Appendix-1

Integration Flow

Integrated Deployment Scenario

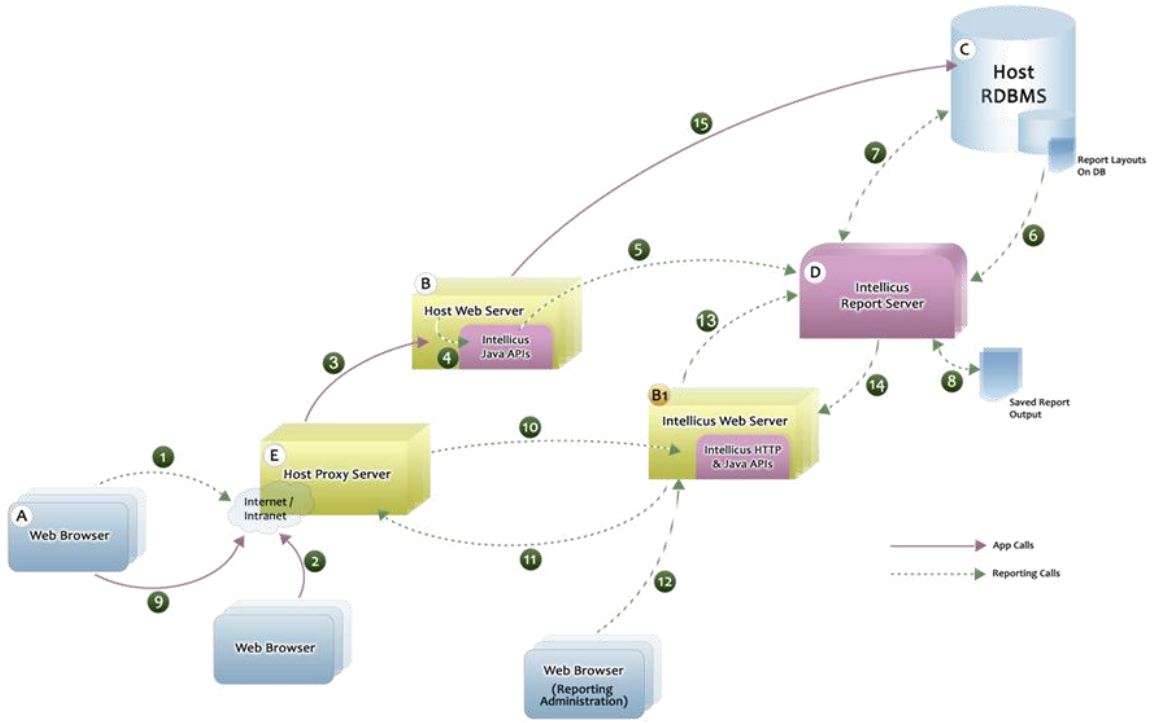


Fig. Integrating Intellicus

Single Sign-on

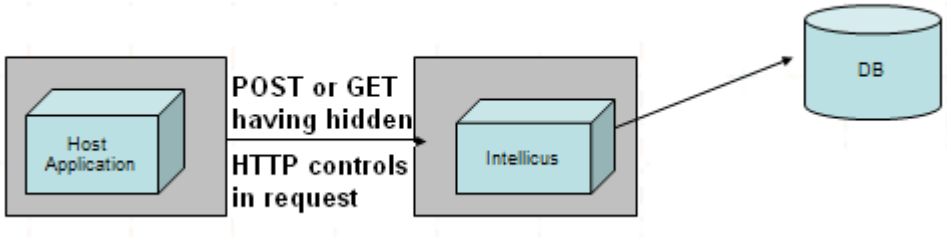
To achieve Single Sign-on, the following need to be considered:

1. **Synchronizing users**
   - Intellicus provides JAVA/HTTP APIs to manage users from a host application
   - You can choose to:
     - Create/Edit users in Intellicus when users are created/ edited in host application OR
     - Check and Create/Edit users in Intellicus whenever a user logs in to host application
   - User Mapping
     - The users in Intellicus shall be shadow users (without storing their passwords in Intellicus system)
     - Each user will map one-to-one with host application user

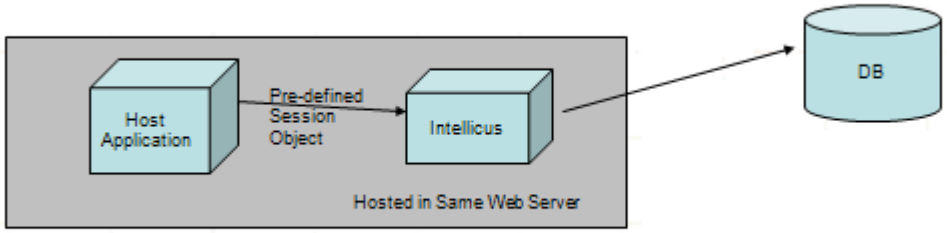
2. Passing Credentials when navigating to Reports

When user navigates from host application to Intellicus the credentials can be passed as:

- Hidden HTTP Parameters



- Session Variables (Only in case of same web server)

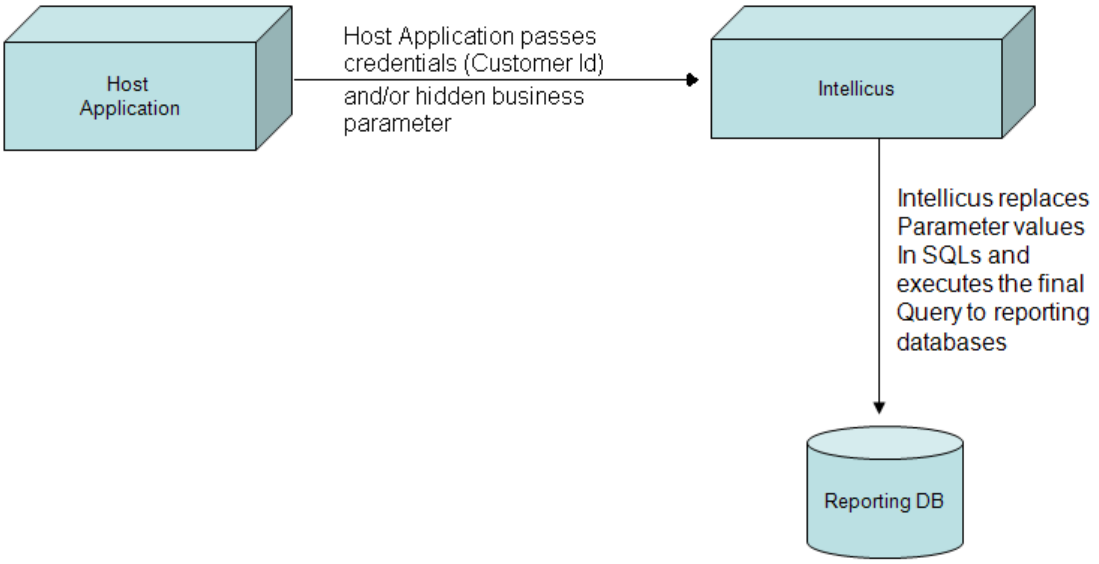


3. Authentication

- Following mechanisms can be used to verify passed credentials again in Intellicus
- Call Back
  - You can Plug-in your custom code to authenticate passed Userid (loginname), Password (password), Orgid(Customerid) from your database.
- External authenticators
  - You can configure external authenticators such as LDAP

Data Filtering

- UserId and CustomerId are now available with Intellicus as passed from host application
- Data filtering at User level or Customer level can be achieved by:
  - Each Report SQL will be designed using parameters (place holders) in the WHERE clause to filter data
    Example: (Select … where Tab1.CustomerID=**<%CustomerId%>**
- Intellicus Report Server will replace the parameter values just before SQL execution



Intellicus Repository

Intellicus repository is a set of few tables in database that stores:

- Report Categories and Reports
- Users and Roles
- Schedules
- Access Rights of users on reports
- Intellicus repository also uses file system to store generated report output, which are saved for future use

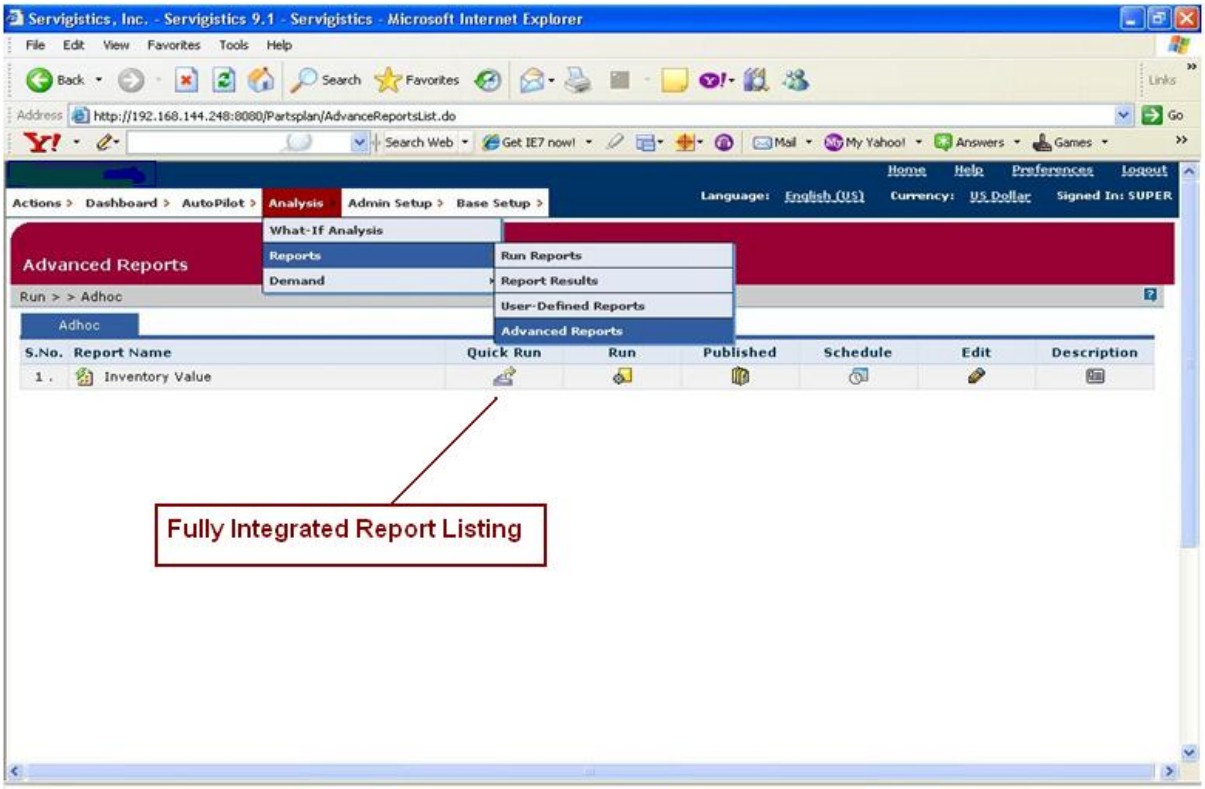
Access Rights

- Intellicus manages Users and Roles under Organizations
  - An Organization logically maps to a Customer of yours
- Roles can be granted to users
- Access Rights can be assigned to Users and Roles for
  - Listing and Running Reports in selected Report Categories

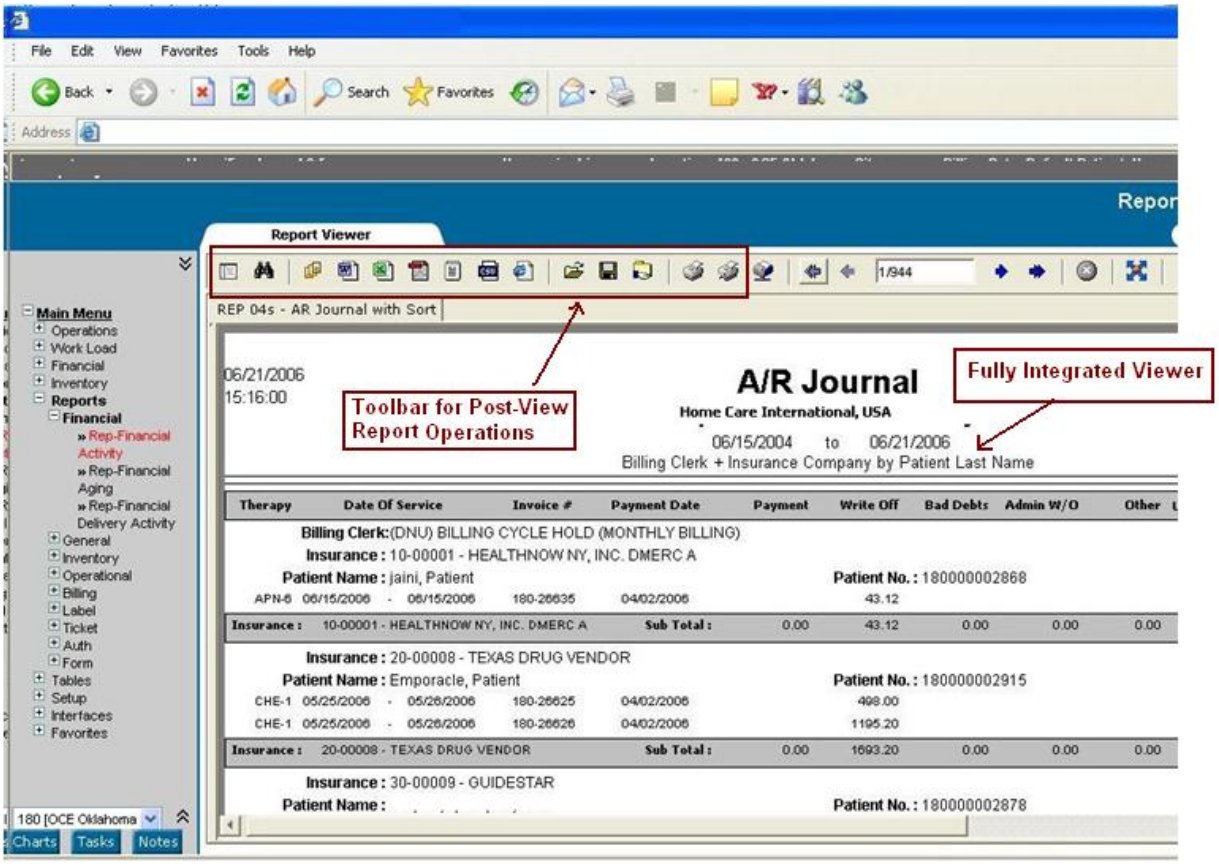
Embedding Report Screens

- Each Intellicus screen is an embeddable web page
  - Any screen can be embedded into your web application seamlessly using URLs in iFrames
  - You can also choose to call complete Intellicus portal in new window (bypass login screen)
  - Intellicus screens can be customized using style sheets to match your web application's UI standards
- It is also possible to navigate back to your application's screens from reports by designing report drilldowns accordingly

Embedding Report List SCreen



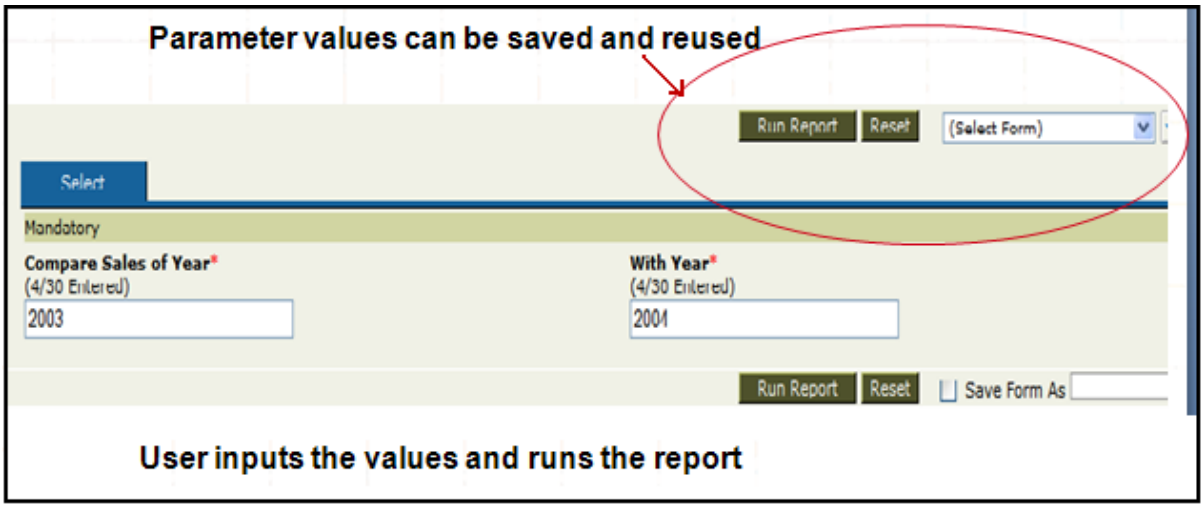
Embedded Report Screen



Reporting Flow

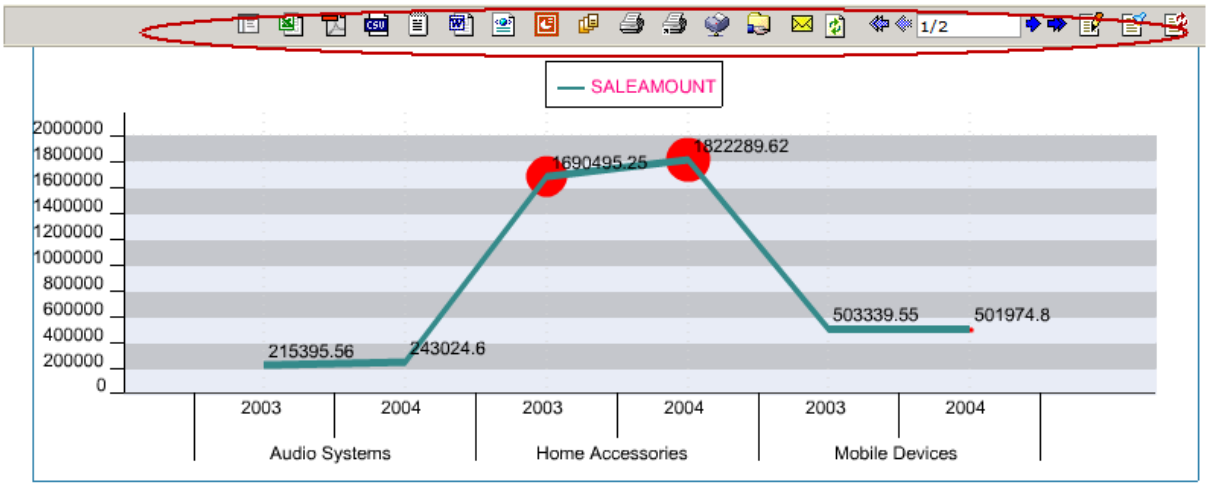
- Host application can call URL to list report categories
  - Intellicus will list categories accessible to logged in user
- User selects a report category to list reports and gets options to
  - Run a report
  - Schedule a report
  - Email a Report
  - View saved reports instance

Prompt Report Parameters Screen



Report Output Screen

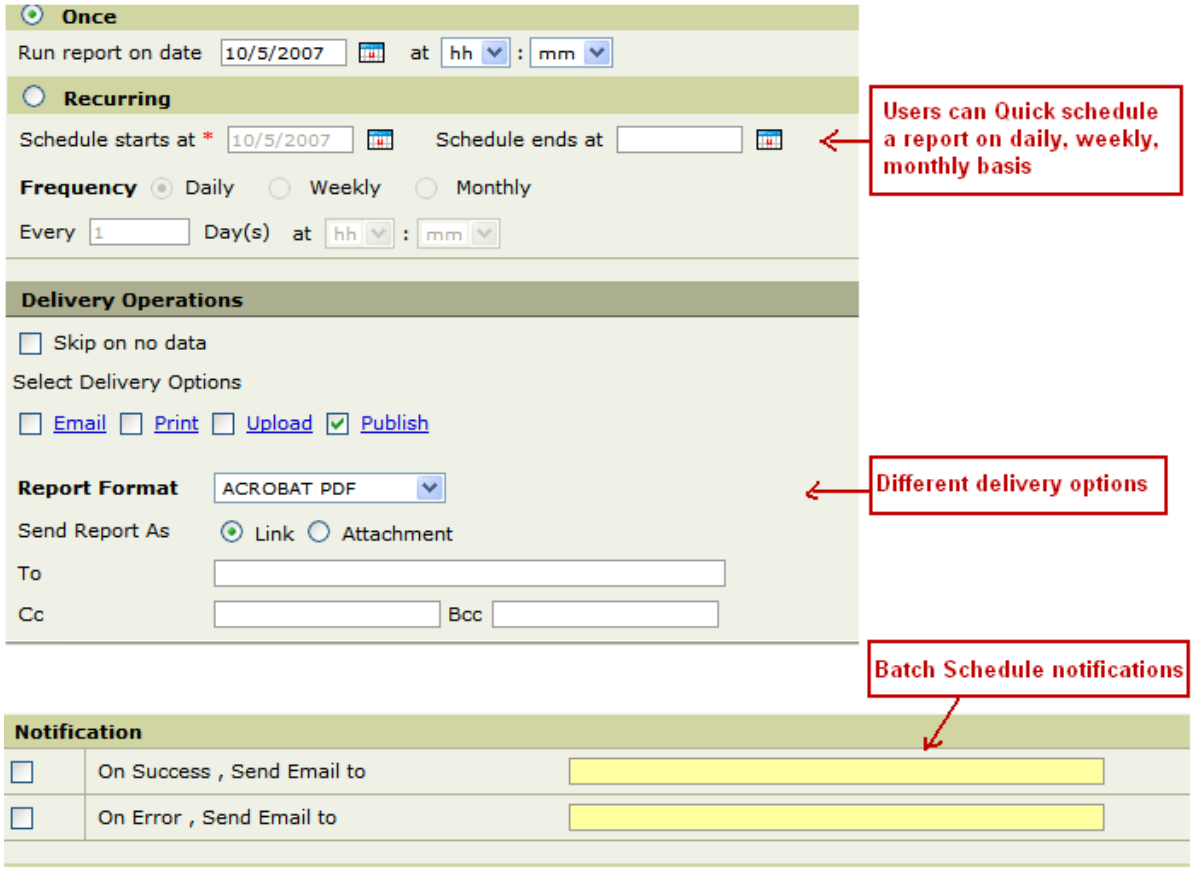
Different operations that can be performed from report output (save, email, export in different formats etc.)



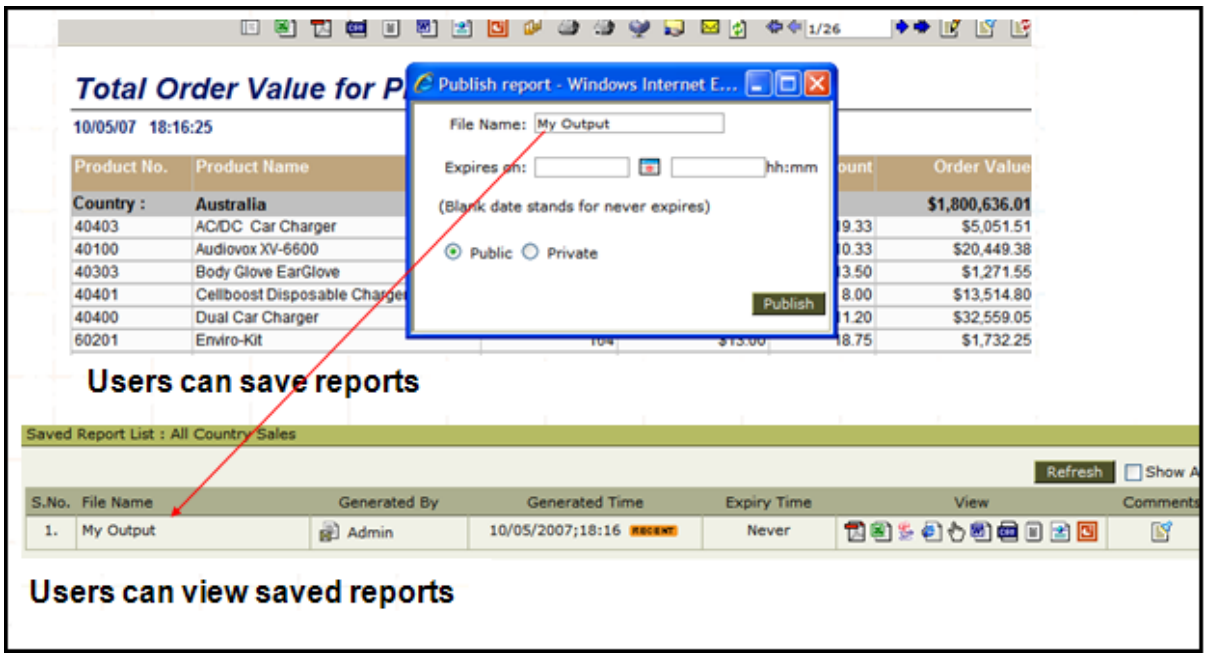
Total Sales for year	2003	$2,409,230.36
Product		**Total Sales Amount**
Audio Systems		**$215,395.56**
Gmini XS200		$123,128.64

Total Sales for year	2004	$2,567,289.02
Product		**Total Sales Amount**
Audio Systems		**$243,024.60**
Gmini XS200		$116,112.00

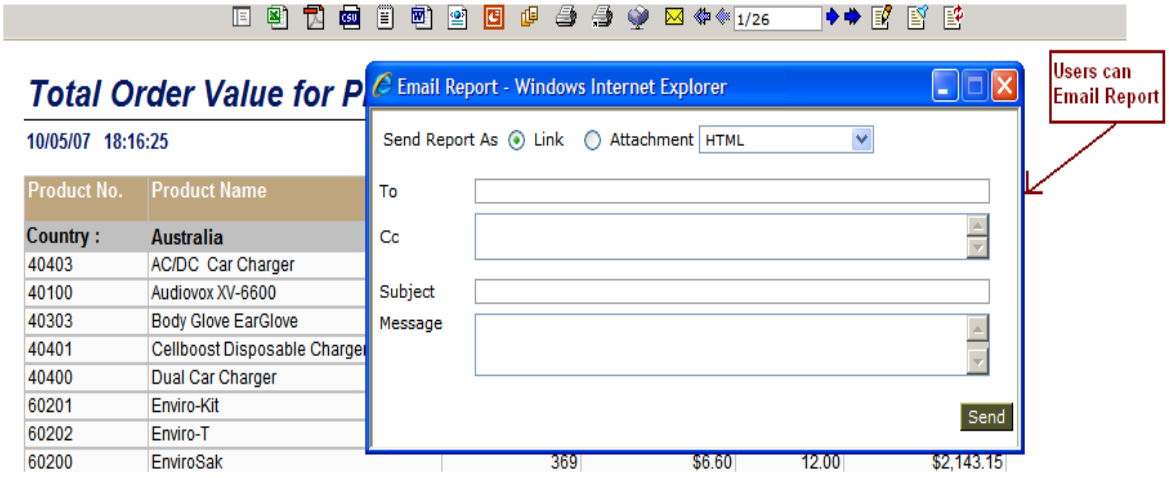
Report Scheduling Screen



Saving Report Output Screen



Report Emailing Screen



Appendix-2

Report Execution RESTful API

Below is the code snippet to call RESTful API for Report Execution using GET and POST methods. This RESTful API is used to get the Streamed report output which can then be saved in a desired file type.

Import

```
//I/O imports
import java.io.BufferedWriter;
import java.io.FileOutputStream;
import java.io.OutputStream;
import java.io.OutputStreamWriter;
import java.io.InputStream;
import java.io.UnsupportedEncodingException;

//URL Connection related imports
import java.net.HttpURLConnection;
import java.net.URL;
import java.net.URLConnection;
```

This Rest API supports User credentials to be passed in request directly or passing Intellicus token (generated using Single Sign On) to perform the required operation.
GET method only supports intellicus token authentication though POST method allows User to pass User Credentials i.e.

- USER_CREDENTIAL_1 -> UserId
- USER_CREDENTIAL_2 -> Password
- USER_CREDENTIAL_3 -> Organization Id

Steps Using GET Method:

1. Single Sign On should be mandatorily implemented for authenticating users that are present in Organizations whose authentication check is performed by Host Application. Here, password is not required to be passed.

   Client sends request to Intellicus for authentication and in return, gets token on successful authentication. This token can be used for further Reporting Requests so as to identify user by Intellicus.

   Place intellicaSSO.jar file in the classpath of your application
   This jar file is located at: <Intellicus_Install_Path>\Intellicus\APIs

```
//Single Sign on imports
import com.intellicus.integration.singlesignon.Enums;
import com.intellicus.integration.singlesignon.SingleSignOn;
import
com.intellicus.integration.singlesignon.SingleSignOnException;
```

```
//Creating UserInfo object using requester user credentials
UserInfo userInfo=new UserInfo(userId, orgId);

singleSignOn.setUserInfo(userInfo);

// Set the path for Intellicus Web application
singleSignOn.setIntellicusURL("http://<IP>:<PORT>/intellicus");

//call getIntellicusToken().
//this method returns a intellicus token string, if user
authentication is done successfully.
intellicusToken=singleSignOn.getIntellicusToken();
```

Please refer IntellicusSingleSign-on.doc for more details on single sign on.

Note: IntellicusSingleSign-On.doc will be provided with Intellicus setup. Path: <Intellicus_Install_Path>\Docs\Manuals

2. Pass intellicus token and other Report parameters to call REST API.

```
String url =
http://<IP>:<POrt>/rest/ReportExec/query?REPORT_ID=E00B3E24-9034-
D022-134F-53B19687C8A4&REPORT_FORMAT=xls&SYS_ZIPPED=FALSE&
&INTELLICUS_TOKEN="+intellicusToken+"&prmCustomerId=1009,1010&prm
Customer=SalesCust";
```

3. Open HTTP connection to request Rest API for Report Execution and get Report Output in desired format

```
URL url = new URL(url);
URLConnection conn = url.openConnection();
HttpURLConnection httpConn;
httpConn = (HttpURLConnection)conn;

// opens input stream from the HTTP connection
InputStream inputStream = httpConn.getInputStream();
```

```
String saveFilePath = "C:\\SalesReport.xls";

// opens an output stream to save into file
FileOutputStream outputStream = new FileOutputStream
(saveFilePath);

byte[] buffer = new byte[1024];
int bytesRead = 0;
while ((bytesRead = inputStream.read(buffer)) != -1) {
  outputStream.write(buffer, 0, bytesRead);
  outputStream.flush();
}
outputStream.close();
inputStream.close();
```

Steps Using POST Method:

1. URL for calling Rest service using POSt method-

```
  http://<Intellicus_IP>:<Port>/intellicus/rest/ReportExec
```

2. Client can directly send User Parameters in request in case of calling Rest Service using POST method. (Apart from using Single Sign On)

Set the form parameters as below and submit to above URL using POST method-

```
  <html><body>
  <form  action="http://localhost:8081/intellicus/rest/ReportExec"
method="POST">
  <input type="hidden" name="USER_Credential_1" value="Admin"/>
  <input type="hidden" name="USER_Credential_2" value="Admin"/>
  <input type="hidden" name="USER_Credential_3"
value="Intellica"/>
  <input  type="hidden"  name="REPORT_ID"  value="4C569ED9-66AE-
74DD-E825-56BE01929899"/>
  <input type="hidden" name="REPORT_FORMAT" value="xls"/>
  <input type="hidden" name="INTELLICUS_TOKEN" value=""/>
  <input type="hidden" name="prmCustomerId" value="1005,1010"/>
  <input type="hidden" name="prmCustomer" value="SalesCust"/>
  <input type="submit">Run Report</input>
  </form>
  </body></html>
```

Steps for implementing Single Sign On are same as of GET method.