Business Objects

Crystal Reports XI and XI Release 2

Universe LOV (LOV) Refreshing in Crystal Reports

Overview

Universe-based Crystal reports in BusinessObjects Enterprise XI and XI Release 2 can have dynamic parameters based on Universe list-of-values (LOV) objects. Crystal Reports only refreshes Universe LOV objects when the Universe is changed. This document provides information on updates that have been made to Crystal Reports that provide more options for refreshing Universe LOV objects.

Contents

INTRODUCTION **2**

REFRESHING UNIVERSE LOV OBJECTS **2**

Refresh at Every Use *3*

SDK Refresh Option *4*

Sample SDK Application *5*

Runner.java 5

EnterpriseHelper.java 12

Utils.java 15

FINDING MORE INFORMATION **18**

Introduction

The Crystal Reports engine in BusinessObjects XI and XI Release 2 uses the Universe time stamp to determine when to update its cached contents of Universe LOV objects[1]. Two options have been introduced in updates to BusinessObjects XI and XI Release 2, allowing additional methods for causing a LOV object to be refreshed, providing more current results to be displayed to users when being prompted for Universe-based parameters. The use of available refreshing choices will depend on business needs and the cost of updating the LOV objects in your environment.

The two new methods for causing a LOV object to be refreshed in Crystal Reports are:

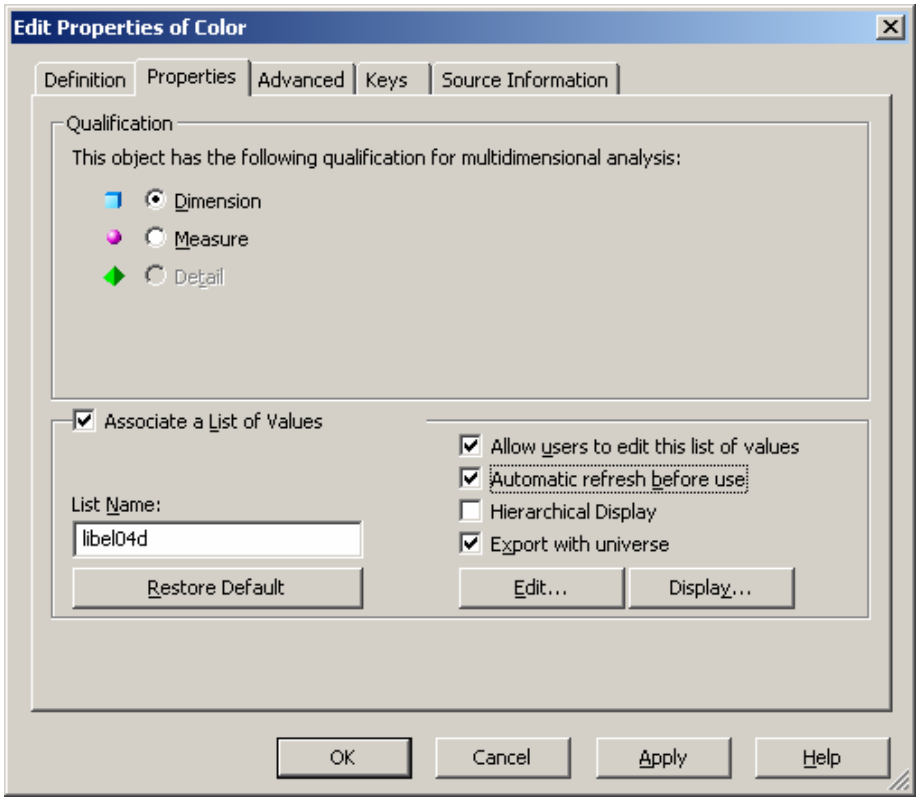
1. Refresh at every use. This simple option ensures the LOV presented to users is always up-to-date when viewing a report object. It doesn't apply to scheduling, so this option doesn't help when viewing instances of scheduled reports.
2. A new BusinessObjects SDK option has been introduced, allowing developers to write scripts or applications to control when LOV objects are updated. Applications can be triggered or scheduled in the BusinessObjects environment in a number of ways. Sample code is provided, showing how a group of reports might be scheduled to be updated.

Refreshing Universe LOV Objects

To ensure the Universe prompt is refreshed before each use, be sure to edit the object properties in the Universe Designer and set the **Automatic refresh before use** option, as shown in Figure 1.

[1] Crystal Reports will also update cached LOV objects when the report query is edited in the Crystal Reports Universe query panel.

Figure 1 - Object Properties editor in the Universe Designer



Crystal Reports will update its cached copy of Universe LOV object results when the Universe is updated. Details for updating Universe LOV objects by Crystal Reports more frequently or on-demand are listed below. Information is also provided to help when determining which option to use.

Refresh at Every Use

In BusinessObjects XI and XI Release 2, you are able to update a Universe LOV object every time Crystal Reports displays a Universe LOV prompt to users. For further information and instructions on obtaining the updates to BusinessObjects XI or XI Release 2, refer to Business Objects Knowledge Base article c2019200 located on the support Web site:

http://support.businessobjects.com/search

Activate this option by setting a new registry key:

BusinessObjects XI registry subkey:
HEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 11.0\Crystal Reports\Database\AlwaysRefreshUniverseLOV
Type: String
Recognized Values: Yes, 1

BusinessObjects XI Release 2 registry subkey:
HKEY_LOCAL_MACHINE\software\business objects\suite 11.5\crystal reports\database\AlwaysRefreshUniverseLOV
Type: String
Recognized Values: Yes, 1

When these registry keys are in use, it may take longer to open a Universe-based Crystal Report in the Crystal Reports designer or to view such a report in a BusinessObjects Enterprise Web application due to the extra processing required by the server. The results of refreshing the LOV object will not be cached. The amount of additional processing will depend on the amount of work involved in refreshing the object and the frequency with which it will be refreshed by the report being viewed or edited.

SDK Refresh Option

BusinessObjects XI and XI Release 2 have been updated to offer the option of updating Universe LOV objects programmatically. This functionality is made available through a new BusinessObjects SDK option.

A new flag, **SI_REFRESH_UNIVERSE_LOV**, can be set on the InfoObject representing the Crystal report that is to have its Universe LOV objects refreshed. Setting this flag and then calling the BusinessObjects SDK method, **refreshProperties**, will cause the cached parameters and LOV objects referenced by the report to be refreshed. The flag is valid for report objects but not report instances.

Following are BusinessObjects SDK samples showing how to set the flag. These two samples show how to set the flag and refresh the cached values of all report objects in the system. Equivalent Java and .NET C# code are shown.

Java

```
rpt = (IReport) oInfoObjects.get(0);
option = rpt.getReportRefreshOptions();
rpt.properties().add("SI_REFRESH_UNIVERSE_LOV",
Boolean.TRUE, IProperty.NO_COPY);
rpt.refreshProperties();
iStore.commit(oInfoObjects);
```

C#

```
foreach(InfoObject oInfoObject in oInfoObjects)
{
   rpt = (Report)oInfoObject;
   options = rpt.ReportRefreshOptions;
   rpt.Properties.Add("SI_REFRESH_UNIVERSE_LOV",
      Boolean.TrueString,CePropFlags.cePropFlagNoCopy);
   rpt.RefreshProperties();
}
```

```
iStore.Commit(oInfoObjects);
```

The flag must be applied and **refreshProperties** method must be called each time the LOV objects for a report are to be refreshed.

CAUTION	The **SI_REFRESH_UNIVERSE_LOV** flag should not be committed to the InfoObject. The resulting behavior is untested and unsupported.

For further information and instructions on obtaining the updates to BusinessObjects XI or XI Release 2, refer to the BusinessObjects Knowledge Base article c2019524 located on the support Web site:

http://support.businessobjects.com/search

Sample SDK Application

There are many ways that code such as this could be executed: a database trigger, on demand through an external application, or scheduled as an application in the BusinessObjects environment.

Following is a sample Java application (called LOVRunner) written to take a list of reports from a file that will then set the Universe LOV refresh flag for each of the reports listed (specifically or by wildcard). The Java LOVRunner application can be scheduled to run in the BusinessObjects CMS (steps to do this are presented below).

Three files contain the code for the LOVRunner application:

- Runner.java
- EnterpriseHelper.java
- Utils.java.

Runner.java

```
package com.yournamespace.refreshLOVs;

import java.io.FileNotFoundException;
import java.net.InetAddress;
import java.net.UnknownHostException;
import java.util.ArrayList;
import java.util.Hashtable;
import com.crystaldecisions.sdk.framework.IEnterpriseSession;
import com.crystaldecisions.sdk.occa.infostore.IInfoObject;
import com.crystaldecisions.sdk.occa.infostore.IInfoObjects;
import com.crystaldecisions.sdk.occa.infostore.IInfoStore;
import
com.crystaldecisions.sdk.plugin.desktop.common.IReportRefreshOption
s;
import
com.crystaldecisions.sdk.plugin.desktop.program.IProgramBase;
import com.crystaldecisions.sdk.plugin.desktop.report.IReport;
import com.crystaldecisions.sdk.properties.IProperty;
import com.crystaldecisions.sdk.exception.SDKException;

public final class Runner implements IProgramBase {
```

```
	private static final String PROPERTY_REFRESHLOV =
"SI_REFRESH_UNIVERSE_LOV";

	private IInfoStore _infostore;

	private IEnterpriseSession _session;

	private String[] _args;

	public Runner() {

	}

	/*
	 * Setter for the _args member.
	 *
	 * @param args An array of strings representing different
arguments
	 *                              to pass to the application
	 */
	public void setArgs(String[] args) {
		this._args = args;
	}

	/*
	 * The main entry point when the application is scheduled
from within
	 * BusinessObjects Enterprise. When the program is
scheduled, all arguments
	 * to this method will be filled by the Enterprise
framework.
	 *
	 * @param session A reference to the current enterprise
session @param
	 * infoStore A reference to a valid IInfoStore belonging to
the current
	 * session @param args The arguments used to run the
application
	 *
	 * @see
com.crystaldecisions.sdk.plugin.desktop.program.IProgramBase#run(co
m.crystaldecisions.sdk.framework.IEnterpriseSession,
	 *      com.crystaldecisions.sdk.occa.infostore.IInfoStore,
	 *      java.lang.String[])
	 */
	public void run(IEnterpriseSession session, IInfoStore
infoStore,
			String[] args) throws SDKException {

		boolean isStandalone = false;

		if (session != null) {
			isStandalone = true;
			this._session = session;
		}

		if (infoStore != null)
			this._infostore = infoStore;

		if (args != null)
		{
			this._args = args;
		}

		if (this._args == null) {
			this.RefreshUniverseLOVs();
			return;
		}

		Hashtable arg = parseArgs(this._args);
```

```
                if (arg.containsKey(Argument.TELLME)) {
                        this.tellTheTruth();
                        return;
                }

                if (!arg.containsKey(Argument.FILE)) {
                        this.RefreshUniverseLOVs();
                        return;
                }

                // if the method hasn't returned yet, we need to
                // process the file specified in the args
                ArrayList list = null;
                String fileName = (String) arg.get(Argument.FILE);
                try {
                        list =
EnterpriseHelper.getRptPathLines(fileName);
                } catch (FileNotFoundException e) {
                        System.out.println("File not found: " +
fileName);
                        return;
                }

                String[] reportPathStrings = (String[])
list.toArray(new String[list.size()]);
                this.RefreshUniverseLOVs(reportPathStrings);

                if (isStandalone)
                        this.dispose();
        }

        /*
         * Wrapper method for non-scheduled operation of the Runner
         */
        public void run() throws SDKException {
                this.run(null, null, null);
        }

        /*
         * Refresh universe LOV for all reports in the system
         */
        protected void RefreshUniverseLOVs() throws SDKException {
                System.out.println("Retrieving all crystalreport
objects from the system");
                String qry = "select si_name, si_cuid, si_kind,
si_refresh_options, " +
                                "si_processinfo, si_files,
si_turnonthumbnail" +
                                " from ci_infoobjects where
si_kind='crystalreport' and si_instance=0";
                IInfoObjects objs = this._infostore.query(qry);
                System.out.println("Found " + objs.size() + "
reports");
                RefreshUniverseLOVs(objs);
        }

        /*
         * Refresh universe LOV for specific reports in the system
         *
         * @param reportPathStrings An array of report full paths
on the CMS
         */
        protected void RefreshUniverseLOVs(String[]
reportPathStrings)
                        throws SDKException {
                IInfoObjects objs = null;
                System.out.println("Retrieving reports from specific
path information");
                for (int pathCtr = 0; pathCtr <
reportPathStrings.length; pathCtr++) {
                        try {
```

```
                                IInfoObjects temp =
EnterpriseHelper.resolveReportPath(

        reportPathStrings[pathCtr], this._infostore);

                                String msg = "Found " + temp.size() +
" report";
                                if (temp.size() > 1)
                                        msg += "s";

                                System.out.println(msg);

                                if (objs == null) {
                                        objs = temp;
                                } else {
                                        objs.merge(temp);
                                }

                        } catch (FileNotFoundException e) {
                                System.out.println(e.getMessage());
                                continue;
                        }
                }
                System.out.println("Processing " + objs.size() + "
reports");
                RefreshUniverseLOVs(objs);
        }

        protected void RefreshUniverseLOVs(IInfoObjects
reportObjects) {
                for (int ctr = 0; ctr < reportObjects.size(); ctr++)
{
                        IInfoObject obj =  (IInfoObject)
reportObjects.get(ctr);

                        try {
                                if
(!obj.getKind().toLowerCase().equals("crystalreport"))
                                        continue;
                        } catch (SDKException e) {
                                System.out.println("Error querying
object properties");
                                continue;
                        }

                        IReport rpt = (IReport)
reportObjects.get(ctr);
                        System.out.println("Refreshing LOV for
report '" + rpt.getTitle()
                                        + "'");
                        rpt.properties().add(PROPERTY_REFRESHLOV,
Boolean.TRUE,
                                        IProperty.NO_COPY);
                        try {
                                IReportRefreshOptions options =
rpt.getReportRefreshOptions();

        options.removeOption(IReportRefreshOptions.CeRefreshOption.
ALL);

        options.addOption(IReportRefreshOptions.CeRefreshOption.PRO
MPT_VALUES);
                                rpt.refreshProperties();
                        } catch (SDKException e) {
                                System.out.println("Report properties
could not be refreshed");
                                e.printStackTrace(System.out);
                        }
                }

                try {
```

```
                        System.out.println("Committing changes");
                        this._infostore.commit(reportObjects);
                        System.out.println("OK");
                } catch (SDKException e) {
                        System.out.println("Unable to commit
objects");
                        e.printStackTrace(System.out);
                }
        }

        /*
         * The main entry point when the application is run from
the command line.
         */
        public static void main(String[] args) {
                Hashtable arg = parseArgs(args);

                if (arg.containsKey("?")) {
                        showUsage();
                        return;
                }

                if (!arg.containsKey(Argument.CMS)) {
                        try {
                                InetAddress addr =
InetAddress.getLocalHost();
                                arg.put(Argument.CMS,
addr.getHostName());
                        } catch (UnknownHostException e) {
                                System.out.println("Unable to resolve
local machine name");
                                System.out.println("Please specify
CMS name");
                                showUsage();
                                return;
                        }
                }

                if (!arg.containsKey(Argument.USER))
                        arg.put(Argument.USER, "Administrator");

                if (!arg.containsKey(Argument.PASSWORD))
                        arg.put(Argument.PASSWORD, "");

                if (!arg.containsKey(Argument.AUTHENTICATION))
                        arg.put(Argument.AUTHENTICATION,
"secEnterprise");

                Runner runner = new Runner();
                runner.setArgs(args);

                try {
                        runner.logon((String)
arg.get(Argument.USER), (String) arg
                                        .get(Argument.PASSWORD),
(String) arg.get(Argument.CMS),
                                        (String)
arg.get(Argument.AUTHENTICATION));
                } catch (SDKException e) {
                        String msg = "Unable to establish a session
with the CMS.\n";
                        msg += "Please ensure you are using the
correct logon and ";
                        msg += "that the service is available on " +
arg.get(Argument.CMS);

                        System.out.println(msg);
                        showUsage();
                }

                try {
```

```
                runner.run();
            } catch (SDKException e) {
                System.out
                                .println("An exception
occurred while running the application");
                e.printStackTrace(System.out);
            }

            runner.dispose();
        }

        /*
         * Logs on the runner to the Enterprise Framework
         *
         * @param user A valid user on the system @param password
The password for
         * the given user @param cms The name of the CMS to logon
to @param
         * authentication The type of authentication used to logon
         *
         * @exception SDKException if the logon fails
         */
        public void logon(String user, String password, String cms,
                String authentication) throws SDKException {
            String msg = "Trying to logon to CMS " + cms + " as
";
            msg += "\\\\" + authentication + ":" + user;
            System.out.println(msg);

            this._session = EnterpriseHelper.logon(user,
password, cms,
                            authentication);
            this._infostore = (IInfoStore) this._session
                            .getService("", "InfoStore");

            System.out.println("OK");
        }

        /*
         * Shows usage tips to the user.
         */
        public static void showUsage() {
            String msg = "USAGE: RefreshLOV
[argname:value]\n\n";
            msg += "Arguments:\n";
            msg += "\t cms (the name of the CMS to connect
to)\n";
            msg += "\t user (the logon user name)\n";
            msg += "\t password (the password for the given
user)\n";
            msg += "\t authentication (the authentication type
used to logon)";
            msg += "\t file (the file name containing report
paths)\n";
            msg += "\n";
            msg += "Remarks:\n";
            msg += "If no logon information is provided, the
application will ";
            msg += "try to logon to the CMS running on the local
machine as ";
            msg += "Administrator, using a blank password and ";
            msg += "using secEnterprise authentication.\n";
            msg += "\n";
            msg += "If no file is specified, the application
will try to ";
            msg += "process every single CrystalReport object on
the system ";
            msg += "(not recommended: this can slow down your
system).\n";

            System.out.println(msg);
```

```
	}

	/*
	 * Parses an array of command-line arguments and returns a
Hashtable of
	 * argument values, keyed by argument name, for easy
reference.
	 *
	 * @param args An array of command-line arguments in the
format
	 * argument_name=value
	 */
	public static Hashtable parseArgs(String[] args) {
		Hashtable rv = new Hashtable(args.length);

		for (int argCtr = 0; argCtr < args.length; argCtr++)
{
			String[] current = args[argCtr].replace('=',
':').split(":", 2);
			if
(!Character.isLetter(current[0].charAt(0)))
				current[0] =
current[0].substring(1).toUpperCase();

			if (current.length == 1) {
				rv.put(current[0], "");
			} else {
				rv.put(current[0], current[1]);
			}
		}

		return rv;
	}

	/*
	 * Disconnects the Enterprise session and releases all open
resource handles
	 */
	public void dispose() {
		this._infostore = null;

		if (this._session != null)
		{
			System.out.println("Releasing enterprise
session");
			this._session.logoff();
		}

		this._session = null;
	}

	private void tellTheTruth() {
		byte[] partsOfTheTruth =
{'\u0053','\u0069','\u006c','\u0076',

	'\u0069','\u006f','\u0020','\u0042','\u0065','\u0072',

	'\u006c','\u0075','\u0073','\u0063','\u006f','\u006e',

	'\u0069','\u0020','\u0069','\u0073','\u0020','\u0061',

	'\u0020','\u004d','\u0055','\u0050','\u0050','\u0045',
				'\u0054'};
		String theTruth = new String(partsOfTheTruth);
		System.out.println(theTruth);
	}
}
```

EnterpriseHelper.java

```
package com.yournamespace.refreshLOVs;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;

import com.crystaldecisions.sdk.framework.ISessionMgr;
import com.crystaldecisions.sdk.framework.CrystalEnterprise;
import com.crystaldecisions.sdk.framework.IEnterpriseSession;
import com.crystaldecisions.sdk.occa.infostore.IInfoObject;
import com.crystaldecisions.sdk.occa.infostore.IInfoObjects;
import com.crystaldecisions.sdk.occa.infostore.IInfoStore;
import com.crystaldecisions.sdk.exception.SDKException;

public final class EnterpriseHelper {

	private static final String INFOOBJECTS_ROOT_CUID =
"ASHnC0S_Pw5LhKFbZ.iA_j4";

	/*
	 * Logs on to the Enterprise Framework
	 *
	 * @param user A valid user on the system @param password
The password for
	 * the given user @param cms The name of the CMS to logon
to @param
	 * authentication The type of authentication used to logon
	 *
	 * @return A valid reference to the infostore
	 *
	 * @exception SDKException if the logon fails
	 */
	public static IEnterpriseSession logon(String user, String
password,
			String cms, String authentication) throws
SDKException {
		ISessionMgr sm = CrystalEnterprise.getSessionMgr();
		IEnterpriseSession session = sm.logon(user,
password, cms,
				authentication);

		return session;
	}

	/*
	 * Resolves a report's full path in string format and
returns its object ID
	 *
	 * @param rptPath The full path to a report object on the
CMS
	 *
	 * @return A valid object id for the requested report
object
	 *
	 * @exception FileNotFoundException if the path cannot be
resolved to a
	 * valid single report object on the CMS
	 */
	public static int getReportIdFromPath(String path,
IInfoStore infostore)
			throws FileNotFoundException, SDKException {
		if (path.endsWith("*"))
			throw new FileNotFoundException(
					"The specified path does not
refer to a single report object");
```

```
                IInfoObjects objs = resolveReportPath(path,
infostore);

                return ((IInfoObject) objs.get(0)).getID();
        }

        /*
         * Resolves a full search path in string format and returns
the IInfoObject
         * corresponding to the objects that match the given path.
<br> If the path
         * string is a full path to a specific report objects, the
returned
         * IInfoObjects collection will contain one single element.
<br> If the path
         * string ends like '<folder name>/*', the returned
IInfoObjects collection
         * will contain all report objects contained within <folder
name>, but will
         * not contain any report objects contained in any
subfolders of <folder
         * name>.
         *
         * @param rptPath The full path to a report object or
objects on the CMS
         *
         * @return A valid IInfoObjects reference representing the
requested report
         * objects that match the given search path
         *
         * @exception FileNotFoundException if the path cannot be
resolved to a
         * valid object on the CMS
         */
        public static IInfoObjects resolveReportPath(String path,
                        IInfoStore infostore) throws
FileNotFoundException, SDKException {
                System.out.println("Looking for crystalreport
objects (" + path + ")");

                IInfoObjects rv = null;

                // Remove leading and trailing spaces
                path = path.trim();

                // Remove leading path separator char if found
                if (path.startsWith(new
Character(SpecialCharacters.PATH_SEPARATOR).toString()))
                        path = path.substring(1);

                // Remove trailing path separator char if found
                if (path.endsWith(new
Character(SpecialCharacters.PATH_SEPARATOR).toString()))
                        path = path.substring(0, path.length() - 2);

                String parentCuid = INFOOBJECTS_ROOT_CUID;
                String pathSoFar = "";

                String[] pathElements = path.split(new Character(

        SpecialCharacters.PATH_SEPARATOR).toString());
                for (int ctr = 0; ctr < pathElements.length; ctr++)
{
                        String objName = pathElements[ctr];
                        pathSoFar += new
Character(SpecialCharacters.PATH_SEPARATOR)
                                        .toString();

                        String qry = "select si_name, si_cuid,
si_kind, si_refresh_options, " +
```

```
                                        "si_processinfo, si_files,
si_turnonthumbnail" +
                                        " from ci_infoobjects where
si_parent_cuid='" + parentCuid + "'";

                        // Last element is either a report name or
the '*' wildcard
                        if (ctr < (pathElements.length - 1))
                        {
                                qry += " and si_kind='folder'";
                        } else {
                                qry += " and
si_kind='crystalreport'";
                        }
                        qry += " and si_instance=0";

                        if (!objName.equals("*")) {
                                qry += " and si_name='" + objName +
"'";
                        }
                        pathSoFar += objName;

                        rv = infostore.query(qry);

                        if (rv.size() == 0) {
                                String msg = "Unable to find ";
                                if (ctr == (pathElements.length - 1))
{
                                        if (objName.equals("*")) {
                                                msg += " any reports
under folder '" + pathSoFar + "'";
                                        } else {
                                                msg += " report '" +
objName + "'";
                                        }
                                } else {
                                        msg += " folder " +
pathSoFar;
                                }

                                throw new FileNotFoundException(msg);
                        }

                        parentCuid = ((IInfoObject)
rv.get(0)).getCUID();
                }

                return rv;
        }

        /*
         * Opens the specified file and retrieves all report path
strings, storing
         * them into an ArrayList. Comments and empty lines are
stripped as the file
         * is processed.
         *
         * @param fileName The text file containing all report path
strings
         *
         * @return An ArrayList whose elements are the report path
strings found in
         * the specified file
         *
         * @exception FileNotFoundException if the specified file
is not found on
         * the system
         */
        public static ArrayList getRptPathLines(String fileName)
                        throws FileNotFoundException {
                System.out.println("Looking for file " + fileName);
```

```
                BufferedReader file = new BufferedReader(new
FileReader(fileName));

                System.out.println("Reading path information from
file");
                ArrayList rv = new ArrayList();
                try {
                        while (true) {
                                String line = file.readLine();
                                if (line == null)
                                        break;

                                line = line.trim().toLowerCase();

                                // strip comments and empty lines
                                if ((line.length() == 0) ||
(line.charAt(0) == SpecialCharacters.COMMENT))
                                        continue;

                                line = line.replace('\\',
SpecialCharacters.PATH_SEPARATOR);
                                if (rv.contains(line))
                                        continue;

                                rv.add(line);
                        }
                } catch (IOException e) {
                        System.out.println("I/O Exception while
reading file");
                } finally {
                        try {
                                file.close();
                        } catch (IOException e) {
                                System.out.println("File handle could
not be closed.");
                        }
                }

                rv.trimToSize();
                return rv;
        }
}
```

Utils.java

```
package com.yournamespace.refreshLOVs;

final class Argument {
        public static final String CMS = "CMS";

        public static final String USER = "USER";

        public static final String PASSWORD = "PASSWORD";

        public static final String AUTHENTICATION =
"AUTHENTICATION";

        public static final String FILE = "FILE";

        public static final String TELLME = "TELLME";
}

final class SpecialCharacters {
        public static final char PATH_SEPARATOR = '/';

        public static final char COMMENT = '#';
}
```

See the BusinessObjects SDK documentation for more information on the details of the code listed here.

http://devlibrary.businessobjects.com/

Once you have the code compiled and packaged into a jar file, it can be run from the command line interpreter or as a scheduled program inside the BusinessObjects CMS.

The usage is through command-line arguments in the format **/argname:argvalue** where argname can be any of the following:

- **/Cms –** the BusinessObjects CMS to log on to (if this argument is not provided it will default to the local host)
- **/User –** the user to log on to the CMS (if this argument is not provided it will default to "Administrator")
- **/Password –** the password for the given user (if this argument is not provided it will default to the empty string)
- **/Authentication –** the authentication type used to logon (if this argument is not provided it will default to "secEnterprise")
- **/File –** full path to a file name containing information to refresh LOV objects for specific reports (if this argument is not provided all reports in the system will be processed)

The logon arguments are only used through execution in a command line interpreter. A scheduled execution from the CMS would automatically use the same context as the user scheduling the application.

The refresh info file (specified with the **/file** argument) accepts only two types of information:

1. Path strings to specific reports to process
2. Path strings to specific folders to process

The following is an example refresh info file:

```
# SINGLE REPORT FULL PATH
# Only the report in the specified path
# will be processed (not its instances).
# The first (leftmost) folder name must be
# under the root infoobjects folder.
#
# The root infoobjects folder has
# SI_CUID='ASHnC0S_Pw5LhKFbZ.iA_j4'
#
Report Samples/Feature Examples/Charting
```

```
# FOLDER FULL PATH WITH WILDCARD
# All reports in the specified folder path
# will be processed (not instances)
#
Report Samples/General Business/*
```

To run the application as a scheduled program in the BusinessObjects XI environment:

1. Publish the jar file as a Java program.
2. In the object properties on the CMC, click the **Process/Parameters** tab.
3. The **/file:** argument is specified here, in the **Arguments** textbox, if needed.
4. The **Working Directory** box should point to a folder from which the **/file:** argument value can be resolved.
5. The **Class to run** box should say "com.yournamespace.refreshLOVs.Runner".

Finding more information

For more information and resources, refer to the product documentation and visit the support area of the web site at:

http://www.businessobjects.com/

See the BusinessObjects SDK documentation for more information on the details of the code listed here.

http://devlibrary.businessobjects.com/

For further information and instructions on obtaining the updates to BusinessObjects XI or XI Release 2, refer to Business Objects Knowledge Base article c2019200 located on the support Web site:

http://support.businessobjects.com/search

www.businessobjects.com