Paper 120-2012

# Developing Custom Metadata Reports for SAS® Data Integration Studio

Michael Kilhullen, SAS Institute Inc., Cary, NC

## ABSTRACT

SAS Data Integration Studio 4.x includes new functionality that allows you to run reports through an interactive reporting facility. You can also create custom reports by using SAS Data Integration Studio software's Java-based plug-in functionality. The plug-in can be written to generate report content using SAS code, Java code, or both. This is ideal for developing reports that specifically address the needs of data integration specialists such as documenting jobs or data standards defined in metadata. This paper takes you through the steps necessary to add a basic metadata report and discusses patterns for developing metadata report plug-ins.

## INTRODUCTION

The purpose of the paper is to introduce you to a process for developing metadata reports and adding them to the SAS® Data Integration Studio 4.x Reports interface (Figure 1). I assume that you already know how to extract metadata from the SAS® Metadata Server. If not, the example used in this paper will give you a basic example but you need to spend some time learning this since it can become rather complex. Also, the method used in this paper focuses on PROC METADATA and will require the use of XML maps to transform the extracted metadata to SAS data sets. Finally, the report will be created using standard SAS procedures and Output Delivery System (ODS).

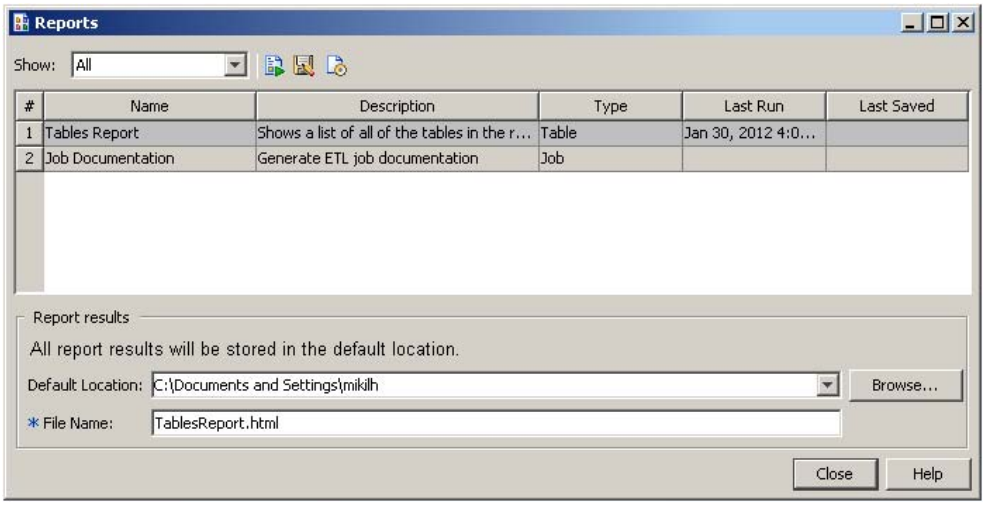


**Figure 1 - Reports Tool Interface in SAS Data Integration Studio**

The Reports tool is designed to easily integrate metadata extraction and reporting through SAS. It provides the ability to change the report type by leveraging ODS. If you design your reports with these constraints, you will be able to quickly add reports to the Reports tool. Although much more complex reporting solutions can be added (for example, the Job Documentation report that is shipped with SAS Data Integration Studio), this is beyond the scope of this paper.

For the purpose of this paper, we will add a simple monitoring report that lists the SASLibrary objects defined in a metadata server and indicates which libraries are using the same directory path on a server.

## DEVELOPING METADATA REPORTS

Extracting metadata can be a complex process depending on metadata that you need for your report. A best practice that I use to initially develop the metadata report is to use Base SAS. This allows me to use the Metadata Browser window to explore metadata, and more easily debug my metadata requests, xml maps, and report code. For this paper, the Base SAS program is listed in Appendix 1: Metadata Extract and Report Program in Base SAS. To help you get started, I will cover some key concepts demonstrated in this code (Figure 2).

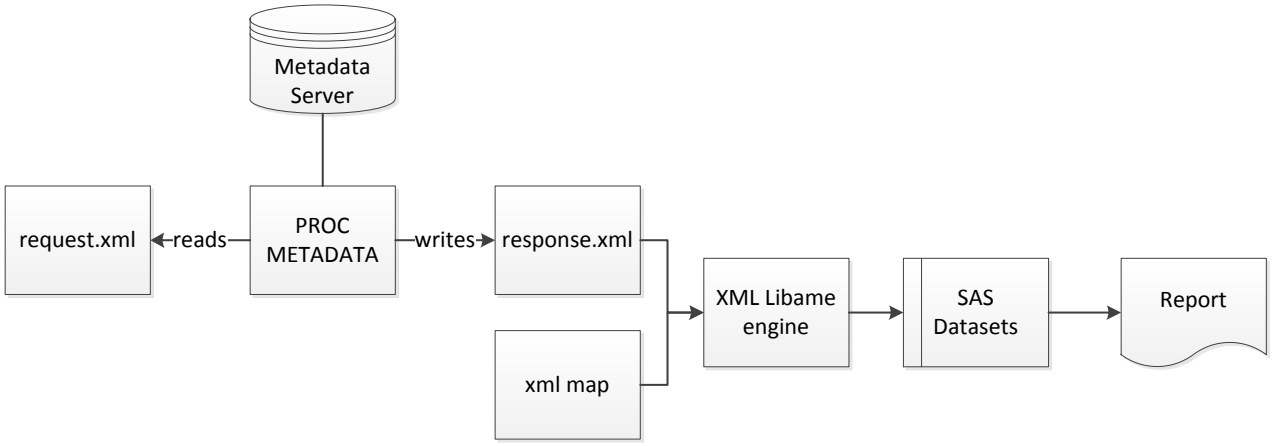


**Figure 2 - Anatomy of a Metadata Report Program Using Base SAS**

SAS metadata reports rely on the use of PROC METADATA to extract the metadata. The description of what metadata needs to be extracted is specified in the request.xml file. When executed, the extracted metadata is added to a response.xml file. We use an XML map to convert the metadata to SAS data sets. Once we have a SAS data set, we can do anything else necessary to massage the data into a report format and produce the final report.

## DETERMINING WHAT METADATA TO EXTRACT

For our report, we need to extract information about SASLibrary objects including information about the directory name and server. To understand how this is represented in metadata, issue the METABROWSE command from your SAS session to open the Metadata Browser. The Metadata Browser allows you to view the metadata server content in a hierarchical view. The Metadata Browser display for a SASLibrary object is shown in Figure 3.

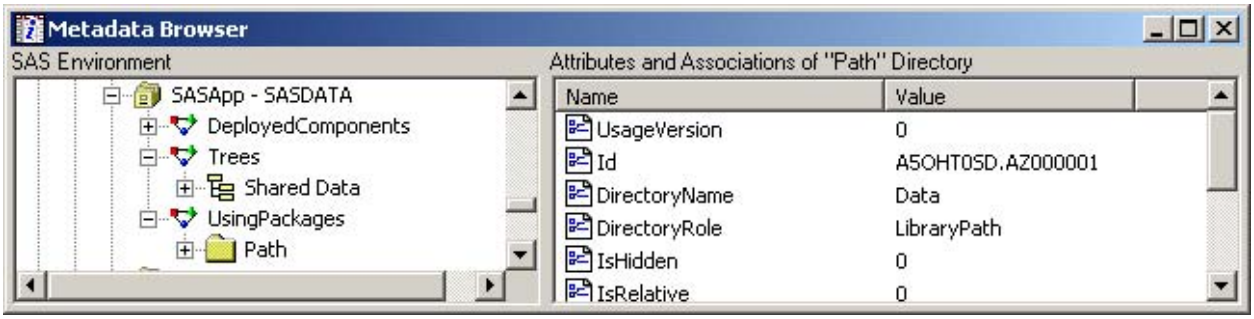


**Figure 3 Metadata Browser View of SASLibrary Object**

If you examine the Metadata Browser window, you will see that a SASLibrary object stores path information that is accessible through the UsingPackages association.

## EXTRACTING METADATA

To extract metadata, we will use PROC METADATA. PROC METADATA reads in an OMI-XML specification and outputs results to the Log window or external file. PROC METADATA allows the XML to be specified using a fileref, so we will adopt a pattern where the input and output are filerefs to code that we generate during execution. This will allow us to modularize the code, make it easier to develop and debug in Base SAS, and facilitate reusability once we transition it into the Java environment. To extract the information that we identified earlier, we will use the following request:

```
<GetMetadataObjects>
<ReposId>$METAREPOSITORY</ReposId>
<Type>SASLibrary</Type>
<Objects/>
<NS>SAS</NS>
<Flags>260</Flags>  <!-- OMI_GET_METADATA(256) + OMI_TEMPLATE(4) -->
<Options>
<Templates>
<SASLibrary ID="" Name="" Desc="" Engine="" isDBMSlibname=""
            Libref="" MetadataCreated="" MetadataUpdated="">
<UsingPackages/>
</SASLibrary>
<Directory ID="" DirectoryName="" IsRelative=""/>
<DataBaseSchema ID="" Name="" Desc="" />
</Templates>
</Options>
</GetMetadataObjects>
```

The <Templates> tag is used frequently in metadata reporting. It allows you to only extract the information needed for the report. For example, I specified that for Directory objects, I only want to see the DirectoryName and IsRelative attributes.

## READING EXTRACTED METADATA

When PROC METADATA runs, it will write the extracted metadata to an XML file. In order to access the content in SAS, XML maps are created. If you do not have experience creating XML maps, you can use the SAS® XML Mapper to visually design the map, and then copy and paste the code it creates into your program. The XML map for the sample program creates a single table containing the information extracted from the metadata server. The use of the SAS XML LIBNAME engine in the sample code is fairly typical so I will not discuss it specifically here. Refer to Appendix 1: Metadata Extract and Report Program in Base SAS for an example of XML map code.

## CREATING THE REPORT

Once the view model is created, you can use your favorite report method in Base SAS to produce the report. Since the example report has only a few columns, PROC PRINT will meet our needs. The final output in SAS is shown in Figure 4.

**SASLibrary Path Information**

| Name | Desc | ServerName | DirectoryName | ALERT |
|---|---|---|---|---|
| Test Study Rawdata | source data library | | | |
| Test Submission submission library | | | | |
| Test Study SDTM | SDTM Library | SASApp | C:\Public | |
| Current Controlled Terminology Library | | SASApp | Data | |
| SASApp - SASDATA | | SASApp | Data | * |
| SDTM Validation Library | | SASApp | Data | * |

**Figure 4 - Metadata Report Results in SAS**

## CONVERTING THE PROGRAM TO A REPORT PLUG-IN

By using Base SAS to develop the metadata report, most of the effort required to create a report plug-in is completed. The next step involves creating the necessary Java classes to plug this code into the SAS Data Integration Studio Reports tool. This process is simplified by using a visual Java development tool such as Eclipse.

## IMPLEMENTING REQUIRED METHODS

An abstract implementation of the reporting interface has been provided called AbstractReport. AbstractReport provides some default implementations of the interface methods. It assumes that the report is being generated with SAS ODS, and the ODS Report Options dialog box is being used.

### Name and Description

The name and description displayed for the report in the Reports tool (Figure 6) is taken from two methods as follows:

```
public String getName() {
   return "Library Report";
}
public String getDescription() {
   return "Generate a report about libraries";
}
```

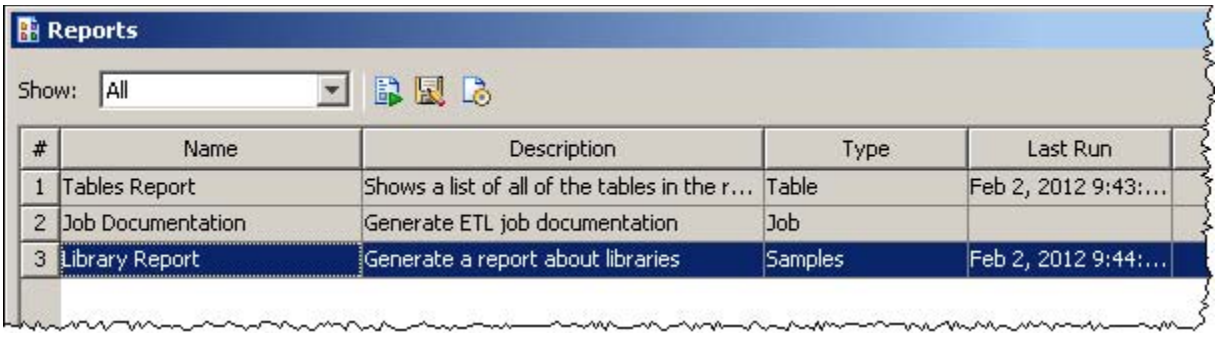


**Figure 6 - Report Displayed in Reports Tool**

### Category

The report category is displayed in the Reports tool under the heading Type. The user can also choose to show reports based on the category name. You can provide the name of an existing category or add a new one. This is especially useful when you are adding reports targeting different user roles. For the sample plug-in, we will create a new category called “Samples” (Figure 5).

```
public String getCategory() {
    return "Samples";
}
```

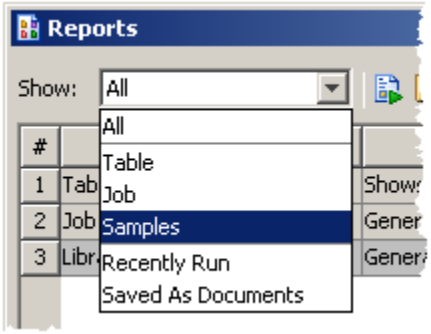


**Figure 5 - Categories Selection**

### Source Code

In the SAS Data Integration Studio environment, the source code for the report must be submitted to the application server. A SASCodeGenerator class is provided in the Java environment that allows you to generate lines of code and/or comments, and to format the code when it is generated. For the most part, the code in the original program can be copied and pasted into the getSourceCode() method, and then each line needs to be enclosed in an addSourceCode() method. A very simple example of using the SASCodeGeneration class is as follows:

```
SASCodeGeneration codeGen = new SASCodeGeneration();
codeGen.addCommentLine("A simple SAS program");
codeGen.addSourceCode("data;\n");
codeGen.indent(2);
codeGen.addSourceCode("x=1;\n");
codeGen.addSourceCode("y=2;\n");
codeGen.undent(2);
codeGen.addSourceCode("run;\n");
```

After your code is added, you can enhance it further using the SASCodeGenerator methods to generate very pretty and well-documented code.

**ODS Output Support**

The AbstractReport class assumes that the report is being generated with SAS ODS, and the ODS Report Options dialog box is being used (Figure 7). In order to leverage this in the generated source code, several methods are available. Prior to generating the source code statements, add this code:

```
codeGen.addSourceCode( "filename myReport \"")
      .addSourceCode( getURL() )
      .addSourceCode("\";\n" );
String sformat = getODSFormatType();
codeGen.addSourceCode( "ods ")
      .addSourceCode( sformat )
      .addSourceCode( " file=myReport \n" );
//Check to see if style sheet is being used
String sStyleSheet = getODSStyleSheet();
//This options only works with HTML...
if ( sformat.equals( ReportingInterface.ODS_HTML ))
      if (sStyleSheet != null && sStyleSheet.length() > 0)
      {
               codeGen.addSourceCode( "stylesheet=(URL=\"file:" )
               .addSourceCode(sStyleSheet.trim())
               .addSourceCode( "\") \n" );
      }
String sAdditionalOptions = getODSAdditionalOptions();
if (sAdditionalOptions != null && sAdditionalOptions.length() > 0)
{
      codeGen.addSourceCode( sAdditionalOptions )
      .addSourceCode( "\n");
}
codeGen.addSourceCode(";\n");
```

This code is fairly standard, so it can be copied and pasted into your report program. After the report statements are generated, we need to close the ODS, so add:

```
codeGen.addSourceCode( "ods ")
.addSourceCode( getODSFormatType() )
.addSourceCode( " close;\n\n" );
```

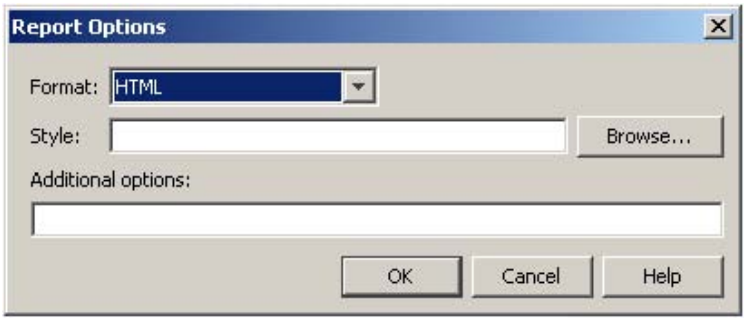


**Figure 7 - ODS Report Options**

## DEPLOYING THE REPORT PLUG-IN

Report plug-ins are made available to SAS Data Integration Studio by creating a manifest, packaging it up into a JAR, and copying it to the SAS Data Integration Studio plug-ins folder. The manifest must contain the Plugin-Init attribute, which specifies the fully qualified class name for your plug-in. The manifest created for the sample plug-in is as follows:

```
Manifest-Version: 1.0
Ant-Version: Apache Ant 1.7.1
Created-By: 17.1-b03 (Sun Microsystems Inc.)
Plugin-Init: com.sas.sample.LibraryReport
```

After creating a JAR, the final step is to copy it to the SAS Data Integration Studio plug-ins folder (for example, c:\program files\SASHome\SASDataIntegrationStudio\4.3\plugins). In order for the plug-in to be picked up when SAS Data Integration Studio starts, it must reside in a subfolder in the plug-ins folder.

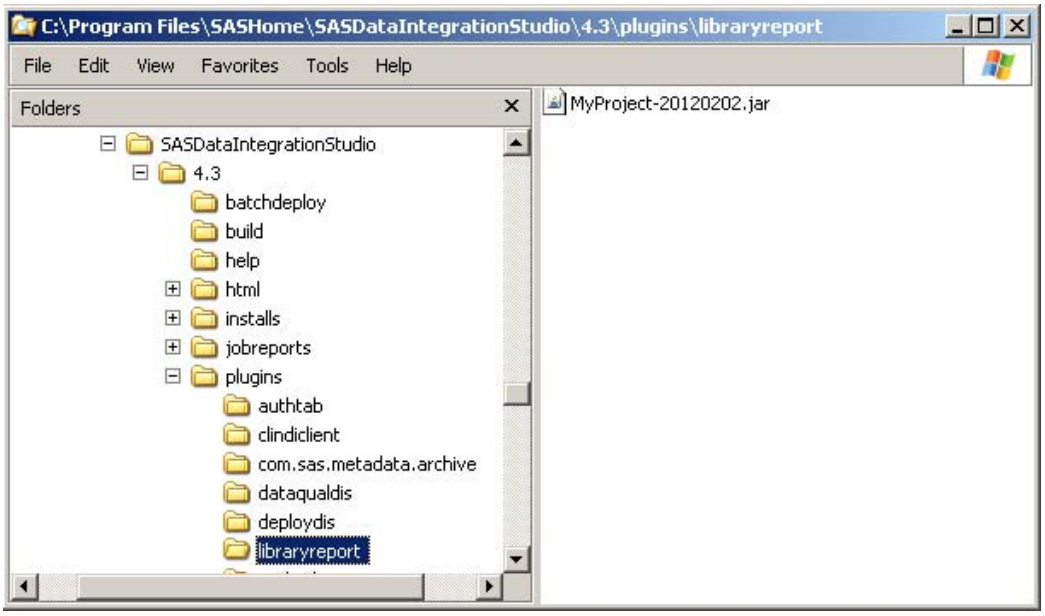


**Figure 8 - JAR Added to a Folder in Plugins**

## RUNNING THE REPORT

After the JAR is deployed, restart SAS Data Integration Studio. Start the Report Tools by selecting the Tools->Reports menu option. If the plug-in loaded correctly, it should appear as a report (Figure 6). After selecting it, you can click the **Run and View a Report** button.

## CONCLUSION

As you can see, by aligning your metadata reporting approach with the design of the AbstractReport class, you can greatly simplify migrating SAS programs to SAS Data Integration Studio plug-ins. By using SAS to develop the metadata report, we are leveraging a more familiar and user-friendly SAS debugging environment to work on the more difficult aspects of metadata reporting, namely extracting metadata and creating XML maps. While the pattern used throughout this paper is applicable to most metadata reporting needs, the ReportInterface also allows you to provide very complex and visual reporting solutions.

## RECOMMENDED READING

SAS Institute Inc. 2011. *SAS® 9.3 Open Metadata Interface: Reference and Usage*. Cary, NC: SAS Institute Inc.

SAS Institute Inc. 2011. *SAS® 9.3 Language Interfaces to Metadata*. Cary, NC: SAS Institute Inc.

SAS Institute Inc. 2011. *SAS® 9.3 Metadata Model: Reference*. Cary, NC: SAS Institute Inc.

SAS Institute Inc. 2011. *SAS® Data Integration Studio 4.3: User's Guide*. Cary, NC: SAS Institute Inc.

SAS Institute Inc. 2011. *SAS® 9.3 XML LIBNAME Engine: User's Guide*. Cary, NC: SAS Institute Inc.

## CONTACT INFORMATION

I would be happy to provide you with the source code used to write this paper as well as an Eclipse Project template that you can use to develop report plug-ins. Your comments and questions are valued and encouraged. Contact the author at:

Michael Kilhullen
SAS Institute
62 Edison Road
Stewartsville, NJ 08886
908-760-6528
michael.kilhullen@sas.com



## APPENDIX 1: METADATA EXTRACT AND REPORT PROGRAM IN BASE SAS

```
/**
 * program that extracts metadata about SASLibrary objects
 * and produces a report that alters user to libraries using the same path
 */

options metaserver="localhost"
metaport=8561
metarepository="Foundation"
metaprotocol="BRIDGE"
metauser="sasadm@saspw"
metapass="{SAS002}A5222D554655B9EC38F4AC360C2FAE72";

/* setup */

filename request temp;
filename response temp;
filename myxmlmap temp;

/* define the model extract */

data _null_;
infile cards missover;
input line $256;
file request;
put _infile_;
cards;
<GetMetadataObjects>
<ReposId>$METAREPOSITORY</ReposId>
<Type>SASLibrary</Type>
<Objects/>
<NS>SAS</NS>
<Flags>260</Flags>  <!-- OMI_GET_METADATA(256) + OMI_TEMPLATE(4) -->
<Options>
```

```
<Templates>
<SASLibrary ID="" Name="" Desc="" Engine="" isDBMSlibname=""
            Libref="" MetadataCreated="" MetadataUpdated="">
<UsingPackages/>
<DeployedComponents/>
</SASLibrary>
<Directory ID="" DirectoryName=""/>
<DataBaseSchema ID="" Name="" Desc="" />
<ServerContext ID="" Name=""/>
</Templates>
</Options>
</GetMetadataObjects>
;
run;

data _null_;
   infile cards missover;
   input line $132.;
   file myxmlmap;
   put _infile_ ;
cards;
<?xml version="1.0" encoding="windows-1252"?>
<SXLEMAP version="1.2" name="SXLEMAP">
<TABLE name="SASLibrary">
<TABLE-PATH syntax="XPath">/GetMetadataObjects/Objects/SASLibrary</TABLE-PATH>
<COLUMN name="Name">
<PATH syntax="XPath">/GetMetadataObjects/Objects/SASLibrary/@Name</PATH>
<TYPE>character</TYPE>
<DATATYPE>string</DATATYPE>
<LENGTH>38</LENGTH>
</COLUMN>
<COLUMN name="Desc">
<PATH syntax="XPath">/GetMetadataObjects/Objects/SASLibrary/@Desc</PATH>
<TYPE>character</TYPE>
<DATATYPE>string</DATATYPE>
<LENGTH>200</LENGTH>
</COLUMN>
<COLUMN name="Libref">
<PATH syntax="XPath">/GetMetadataObjects/Objects/SASLibrary/@Libref</PATH>
<TYPE>character</TYPE>
<DATATYPE>string</DATATYPE>
<LENGTH>8</LENGTH>
</COLUMN>
<COLUMN name="DirectoryName">
<PATH
syntax="XPath">/GetMetadataObjects/Objects/SASLibrary/UsingPackages/Directory/@Dire
ctoryName</PATH>
<TYPE>character</TYPE>
<DATATYPE>string</DATATYPE>
<LENGTH>200</LENGTH>
</COLUMN>
<COLUMN name="ServerName">
<PATH
syntax="XPath">/GetMetadataObjects/Objects/SASLibrary/DeployedComponents/ServerCont
ext/@Name</PATH>
<TYPE>character</TYPE>
<DATATYPE>string</DATATYPE>
<LENGTH>20</LENGTH>
</COLUMN>
</TABLE>
</SXLEMAP>
;
run;
```

```
proc metadata in=request out=response;
run;

libname tmpmeta xml xmlmap=myxmlmap xmlfileref=response access=READONLY;

/* setup a view model to flag dups */

proc sort data=tmpmeta.Saslibrary out=work.sorteddata;
by ServerName DirectoryName Name;
run;

data work.sorteddata;
set work.sorteddata;
length flag $1;
by ServerName DirectoryName;
if not first.DirectoryName then do;
 if DirectoryName ne "" then flag="*";
end;
run;

/* create the view */

proc print data=work.sorteddata label noobs;
title "SASLibrary Path Information";
var Name Desc ServerName DirectoryName flag;
label flag="ALERT";
run;

/* cleanup */

libname tmpmeta;
filename request;
filename response;
```

## APPENDIX 2: JAVA REPORT PLUG-IN BASED ON SAMPLE PROGRAM

```
public class LibraryReport extends AbstractReport {

    @Override
    public String getCategory() {
           return "Samples";
    }

    @Override
    public String getReportingClass() {
           return "com.sas.sample.LibraryReport";
    }

    @Override
    public StringBuffer getSourceCode() {

           SASCodeGeneration codeGen = new SASCodeGeneration();

           codeGen.addCommentLine("Creates a summary report of libraries");
           WsServerRequest svrRequest = ReportingController.getInstance()
                        .getServerRequest();
           WsAppServer appServer = svrRequest.getAppServer();

           if (appServer == null)
                  return new StringBuffer();

           try {
```

```
                codeGen.genMetadataMacrosAndOptions(appServer.getServerContext(),
                        appServer.getServerContext(), true);
            } catch (MdException me) {
                MessageUtil.displayMetadataExceptionMessage(me,
                        MessageUtil.ACCESSING);
            } catch (java.rmi.RemoteException re) {
                com.sas.workspace.Workspace.handleRemoteException(re);
            }
            codeGen.addSourceCode("filename request temp;\n");
            codeGen.addSourceCode("filename response temp;\n");
            codeGen.addSourceCode("filename myxmlmap temp;\n");
            codeGen.addSourceCode("data _null_;\n");
            codeGen.addSourceCode("infile cards missover;\n");
            codeGen.addSourceCode("input line $256;\n");
            codeGen.addSourceCode("file request;\n");
            codeGen.addSourceCode("put _infile_;\n");
            codeGen.addSourceCode("cards;\n");
            codeGen.addSourceCode("<GetMetadataObjects>\n");
            codeGen.addSourceCode("<ReposId>$METAREPOSITORY</ReposId>\n");
            codeGen.addSourceCode("<Type>SASLibrary</Type>\n");
            codeGen.addSourceCode("<Objects/>\n");
            codeGen.addSourceCode("<NS>SAS</NS>\n");
            codeGen.addSourceCode("<Flags>260</Flags>  <!-- OMI_GET_METADATA(256) +
OMI_TEMPLATE(4) -->\n");
            codeGen.addSourceCode("<Options>\n");
            codeGen.addSourceCode("<Templates>\n");
            codeGen.addSourceCode("<SASLibrary ID=\"\" Name=\"\" Desc=\"\"
Engine=\"\" isDBMSlibname=\"\"\n");
            codeGen.addSourceCode("Libref=\"\" MetadataCreated=\"\"
MetadataUpdated=\"\">\n");
            codeGen.addSourceCode("<UsingPackages/>\n");
            codeGen.addSourceCode("<DeployedComponents/>\n");
            codeGen.addSourceCode("</SASLibrary>\n");
            codeGen.addSourceCode("<Directory ID=\"\" DirectoryName=\"\"/>\n");
            codeGen.addSourceCode("<DataBaseSchema ID=\"\" Name=\"\" Desc=\"\"
/>\n");
            codeGen.addSourceCode("<ServerContext ID=\"\" Name=\"\"/>\n");
            codeGen.addSourceCode("</Templates>\n");
            codeGen.addSourceCode("</Options>\n");
            codeGen.addSourceCode("</GetMetadataObjects>\n");
            codeGen.addSourceCode("run;\n");
            codeGen.addSourceCode("data _null_;\n");
            codeGen.addSourceCode("infile cards missover;\n");
            codeGen.addSourceCode("input line $132.;\n");
            codeGen.addSourceCode("file myxmlmap;\n");
            codeGen.addSourceCode("put _infile_ ;\n");
            codeGen.addSourceCode("cards;\n");
            codeGen.addSourceCode("<?xml version=\"1.0\" encoding=\"windows-
1252\"?>\n");
            codeGen.addSourceCode("<SXLEMAP version=\"1.2\" name=\"SXLEMAP\">\n");
            codeGen.addSourceCode("<TABLE name=\"SASLibrary\">\n");
            codeGen.addSourceCode("<TABLE-PATH
syntax=\"XPath\">/GetMetadataObjects/Objects/SASLibrary</TABLE-PATH>\n");
            codeGen.addSourceCode("<COLUMN name=\"Name\">\n");
            codeGen.addSourceCode("<PATH
syntax=\"XPath\">/GetMetadataObjects/Objects/SASLibrary/@Name</PATH>\n");
            codeGen.addSourceCode("<TYPE>character</TYPE>\n");
            codeGen.addSourceCode("<DATATYPE>string</DATATYPE>\n");
            codeGen.addSourceCode("<LENGTH>38</LENGTH>\n");
            codeGen.addSourceCode("</COLUMN>\n");
            codeGen.addSourceCode("<COLUMN name=\"Desc\">\n");
            codeGen.addSourceCode("<PATH
syntax=\"XPath\">/GetMetadataObjects/Objects/SASLibrary/@Desc</PATH>\n");
```

```
            codeGen.addSourceCode("<TYPE>character</TYPE>\n");
            codeGen.addSourceCode("<DATATYPE>string</DATATYPE>\n");
            codeGen.addSourceCode("<LENGTH>200</LENGTH>\n");
            codeGen.addSourceCode("</COLUMN>\n");
            codeGen.addSourceCode("<COLUMN name=\"Libref\">\n");
            codeGen.addSourceCode("<PATH
syntax=\"XPath\">/GetMetadataObjects/Objects/SASLibrary/@Libref</PATH>\n");
            codeGen.addSourceCode("<TYPE>character</TYPE>\n");
            codeGen.addSourceCode("<DATATYPE>string</DATATYPE>\n");
            codeGen.addSourceCode("<LENGTH>8</LENGTH>\n");
            codeGen.addSourceCode("</COLUMN>\n");
            codeGen.addSourceCode("<COLUMN name=\"DirectoryName\">\n");
            codeGen.addSourceCode("<PATH
syntax=\"XPath\">/GetMetadataObjects/Objects/SASLibrary/UsingPackages/Directory/@Di
rectoryName</PATH>\n");
            codeGen.addSourceCode("<TYPE>character</TYPE>\n");
            codeGen.addSourceCode("<DATATYPE>string</DATATYPE>\n");
            codeGen.addSourceCode("<LENGTH>200</LENGTH>\n");
            codeGen.addSourceCode("</COLUMN>\n");
            codeGen.addSourceCode("<COLUMN name=\"ServerName\">\n");
            codeGen.addSourceCode("<PATH
syntax=\"XPath\">/GetMetadataObjects/Objects/SASLibrary/DeployedComponents/ServerCo
ntext/@Name</PATH>\n");
            codeGen.addSourceCode("<TYPE>character</TYPE>\n");
            codeGen.addSourceCode("<DATATYPE>string</DATATYPE>\n");
            codeGen.addSourceCode("<LENGTH>20</LENGTH>\n");
            codeGen.addSourceCode("</COLUMN>\n");
            codeGen.addSourceCode("</TABLE>\n");
            codeGen.addSourceCode("</SXLEMAP>\n");
            codeGen.addSourceCode(";\n");
            codeGen.addSourceCode("run;\n");
            codeGen.addSourceCode("proc metadata in=request out=response;\n");
            codeGen.addSourceCode("run;\n");
            codeGen.addSourceCode("libname tmpmeta xml xmlmap=myxmlmap
xmlfileref=response access=READONLY;\n");
            codeGen.addSourceCode("proc sort data=tmpmeta.Saslibrary
out=work.sorteddata;\n");
            codeGen.addSourceCode("by ServerName DirectoryName Name;\n");
            codeGen.addSourceCode("run;\n");
            codeGen.addSourceCode("data work.sorteddata;\n");
            codeGen.addSourceCode("set work.sorteddata;\n");
            codeGen.addSourceCode("length flag $1;\n");
            codeGen.addSourceCode("by ServerName DirectoryName;\n");
            codeGen.addSourceCode("if not first.DirectoryName then do;\n");
            codeGen.addSourceCode(" if DirectoryName ne \"\" then flag=\"*\";\n");
            codeGen.addSourceCode("end;\n");
            codeGen.addSourceCode("run;\n");

            // ODS: the following block of code is standard for processing the
            // ODS information collected in the Reports Tool
           codeGen.addSourceCode( "filename myReport \"")
           .addSourceCode( getURL() )
           .addSourceCode("\";\n" );

           String sformat = getODSFormatType();
           codeGen.addSourceCode( "ods ")
                  .addSourceCode( sformat )
                  .addSourceCode( " file=myReport \n" );
           //Check to see if style sheet is being used
           String sStyleSheet = getODSStyleSheet();
           //This options only works with HTML...
           if ( sformat.equals( ReportingInterface.ODS_HTML ))
              if (sStyleSheet != null && sStyleSheet.length() > 0)
```

```
            {
               codeGen.addSourceCode( "stylesheet=(URL=\"file:" )
                      .addSourceCode(sStyleSheet.trim())
                      .addSourceCode( "\") \n" );
            }
          String sAdditionalOptions = getODSAdditionalOptions();
          if (sAdditionalOptions != null && sAdditionalOptions.length() > 0)
          {
             codeGen.addSourceCode( sAdditionalOptions )
                    .addSourceCode( "\n");
          }
          codeGen.addSourceCode(";\n");

          // end ODS

           codeGen.addSourceCode("proc print data=work.sorteddata label noobs;\n");
           codeGen.addSourceCode("title \"SASLibrary Path Information\";\n");
           codeGen.addSourceCode("var ;\n");
           codeGen.addSourceCode("label flag=\"ALERT\";\n");
           codeGen.addSourceCode("run;\n");

        codeGen.addSourceCode( "ods ")
        .addSourceCode( getODSFormatType() )
        .addSourceCode( " close;\n\n" );

           codeGen.addSourceCode("libname tmpmeta;\n");
           codeGen.addSourceCode("filename request;\n");
           codeGen.addSourceCode("filename response;\n");

           return codeGen.getSourceBuffer();
    }

    @Override
    public void onSelected() {
           // TODO Auto-generated method stub

    }

    @Override
    public String getDescription() {
           return "Generate a report about libraries";
    }

    @Override
    public String getName() {
           return "Library Report";
    }

}
```