Paper 1257-2017

Let the System Do Repeating Work for You

Laurent de Walick, PW Consulting; Bas Marsman, NN Bank

ABSTRACT

Developing software using agile methodologies has become the common practice in many organizations. We use the SCRUM methodology to prepare, plan and implement changes in our analytics environment.

Preparing for the deployment of new release usually took a two days of creating packages, promoting them, deploying jobs, creating migration script and correcting errors made in the first attempt. A ten working days, two weeks, sprint was effectively reduced to barely seven. By automating this process we were able to reduce the time needed to prepare our deployment to less half a day, increasing the time we can spend developing by 25%.

In this paper we present the process and system prerequisites to automate the deployment process, and process, code and scripts to automate metadata promotion and physical table comparison and update.

INTRODUCTION

In the last decade IT adopted new methodologies like, Agile, SCRUM, DevOps and Continuous Integration, at a rapid pace. The business is now accustomed with these methodologies and expects us to implement new requirements faster and more frequent. Application release automation is a key success factor to deploy new software releases rapid, reliable and responsible.

In SAS® Management Console and SAS® Data Integration Studio many of the tasks involved in release management require manual tasks. Wizards assist the developers and administrators during some of the steps but are still prone to errors. But every change can be registered in the metadata and all metadata can be queried using data step functions and the METADATA procedure. SAS also provides batch tools to export and import metadata and deploy DI Studio jobs from a command prompt.

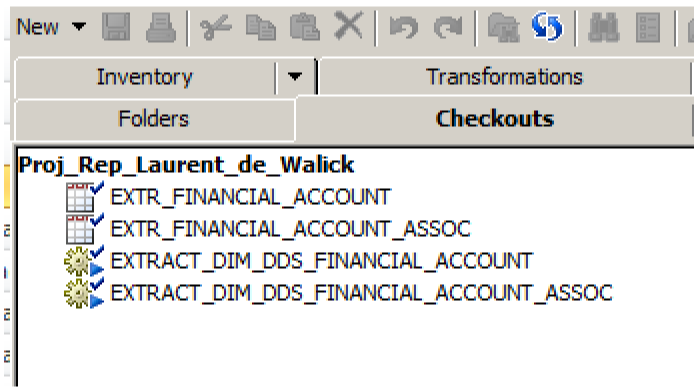
This paper describes what procedures we implemented

- Standard development procedures.
- Generation of export and import batch scripts.
- Generation of SAS scripts to update physical table definition.

The examples in this paper are based on SAS 9.3 installed on a Linux server. The concept should also work in a Windows environment with small modifications.

CHANGE MANAGEMENT

The basis for release automation is to know what metadata objects need to be promoted in a release. The least intrusive method we've found is by using the change management options available in Data Integration Studio. Change management restricts most users from adding or updating metadata in change-managed metadata folders. Developers are authorized to create new metadata objects and check out existing objects into the project repository, as shown in Display 1, which can be checked in into the change-managed metadata folder. Checked out objects are locked in the change-managed folder so that no one else can update them as long as the objects are checked out.

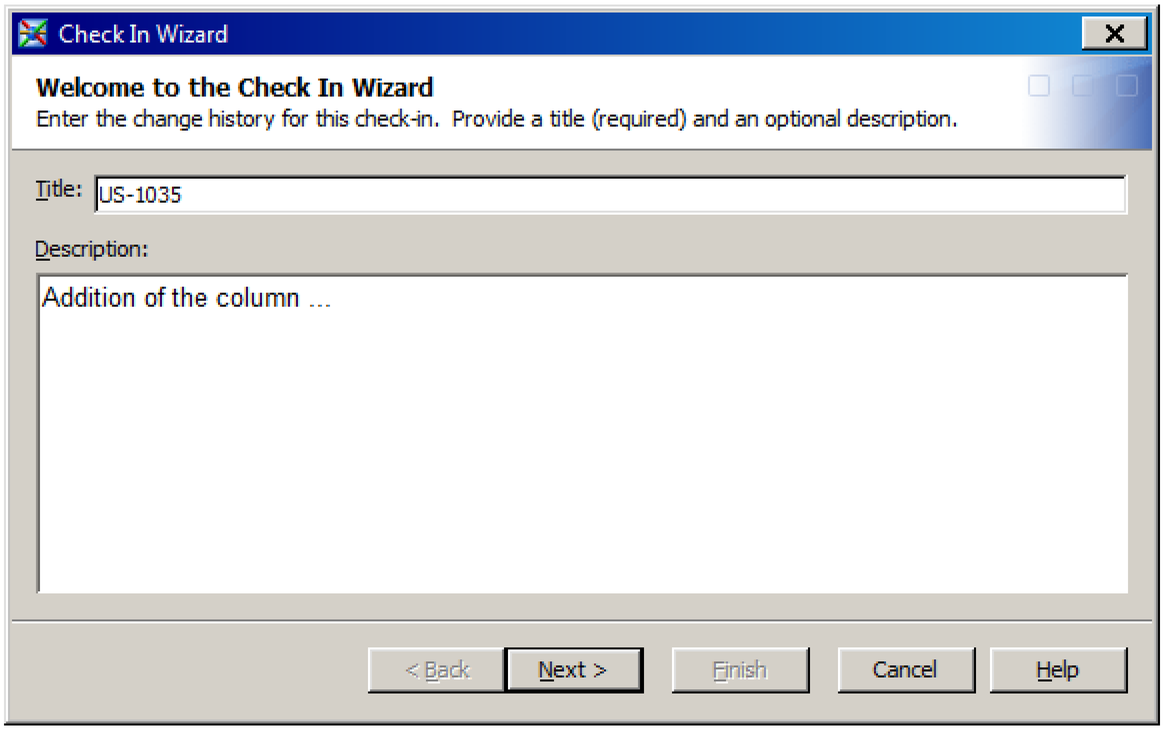


Proj_Rep_Laurent_de_Walick
EXTR_FINANCIAL_ACCOUNT
EXTR_FINANCIAL_ACCOUNT_ASSOC
EXTRACT_DIM_DDS_FINANCIAL_ACCOUNT
EXTRACT_DIM_DDS_FINANCIAL_ACCOUNT_ASSOC

Display 1 Checked out objects in project repository

Once the modifications have been made to objects the developer checks them into the main repository. The check-in wizard asks for a title and a description. It is very important that the right title is entered when checking in, because this is the key the automation scripts will use to select the objects during promotion.

In our case it has to be the exact id of the user story that lead to the change as shown in Display 2. The developer is free to add additional information about the implemented change in the description field.



Welcome to the Check In Wizard
Enter the change history for this check-in. Provide a title (required) and an optional description.

Title: US-1035

Description:
Addition of the column ...

Display 2 Check in wizard ask for title

Using change management not only helps us automating the deployment process, but it also adds insight in who is currently changing specific metadata objects and a history log of when changes were made, who made them and what user story required the change.

GENERATE EXPORT AND IMPORT SCRIPTS

To extract information about changed objects from the metadata we use the generic process as describes in Figure 1. This process will be used a second time to extract flow information from the metadata.

1. First we create a temporary filename reference that will contains the XML for the metadata request. The contents of the file is created using a data step. By adding variables in the data step we create a customized request to only extract information of a single user story.
2. Next we execute PROC METADATA using our generated request XML as input parameter and capturing the output to another temporary filename reference.
3. Then we create an XML map. We choose to add the creation of the XML map in our program and write it to a temporary file. Since it's not dynamic it could also be saved somewhere and reused on every run.
4. The final step is to generate an XML library using the Response XML as source and the XML map.

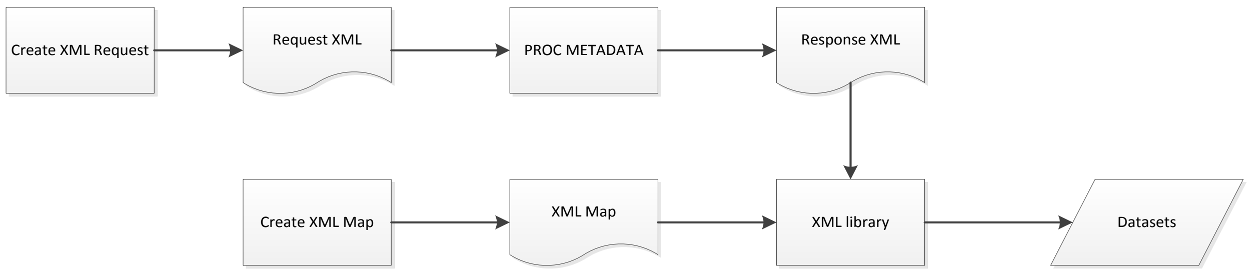


Figure 1 Extract metadata process flow

CREATE XML REQUEST

The following objects are under change management and can be checked in in Data Integration Studio.

- User Defined Transformations
- Libraries
- Tables
- External Files
- Jobs

Deployed Jobs are a special case. It's not possible to deploy jobs from a project repository and as a results it is not possible to select deployed jobs from the change object. By adding the

SAS Data Integration Studio registers check ins from project repositories as *Change* objects in the metadata. The GetMetadataObjects retrieves all objects from a single type. We limit the selection of objects by using the XMLSELECT option and filter on the name attribute containing the name of the user story. The data step to create the request XML is:

```
filename reqObjs temp encoding='utf-8';

data _null_;
 file reqObjs lrecl=32767;
 put '<GetMetadataObjects>';
 put '<Reposid>$METAREPOSITORY</Reposid>';
 put "<Type>Change</Type>";
 put '<Objects/>';
 put '<NS>SAS</NS>';
 put '<Flags>388</Flags>';
 put '<Options>';
 put '<XMLSELECT search="*[@Name ? '''"&release."''']" />';
 put '<Templates>';
```

```
  put '<Change Id="" Name="" MetadataCreated="">';
  put '<Objects />';
  put '</Change>';
  put '<Job Id="" Name="" PublicType="">';
  put '<JFJobs />';
  put '<Trees />';
  put '</Job>';
  put '<JFJob Id="" Name="" PublicType="">';
  …
  put '</Templates>';
  put '</Options>';
  put '</GetMetadataObjects>';
run;
```

Then the request XML is used in the METADATA procedure to perform the request. PROC METADATA returns an XML with the results of the query:

```
filename resObjs temp encoding='utf-8';

proc metadata
 in=reqObjs
 out=resObjs
;
run;
quit;
```

CREATE XML MAP

The next step is to create the XML Map that translates the response XML into usable SAS tables. The easiest method for generation of XML Maps is to use the SAS® XML Mapper. The map the XML Mapper generates is copied into the cards section of a data step. This construct makes the full code portable and reusable with the need for addition files:

```
filename resMap temp encoding='utf-8';

data _null_;
length regel $32767;
file resMap lrecl=32767;
infile cards4 dlm=";";
input regel $;
put regel;
cards4;
<?xml version="1.0" encoding="windows-1252"?>
<SXLEMAP name="responseXml" version="2.1">
<NAMESPACES count="0"/>
<TABLE name="Changes">
 <TABLE-PATH syntax="XPath">/GetMetadataObjects/Objects/Change</TABLE-PATH>
 <COLUMN name="changeId">
  <PATH syntax="XPath">/GetMetadataObjects/Objects/Change/@Id</PATH>
  <TYPE>character</TYPE>
  <DATATYPE>string</DATATYPE>
  <LENGTH>17</LENGTH>
 </COLUMN>
…
</TABLE>
</SXLEMAP>
;;;;
run;
```

FLOW INFORMATION

Deployed flows, just like deployed jobs, are not under change management in Data Integration Studio. To ensure that flows in promotion will be as developed, all flows that can potentially be new or changed are part of the package that will be promoted

The example in Figure 2 shows that a change in Job 4 results in promotion of Sub flow B and the Main flow.

A workaround is needed when a user story's only effective change takes place in a flow. The workaround is to check out and check in a job that is part of the changed flow, without doing any change.

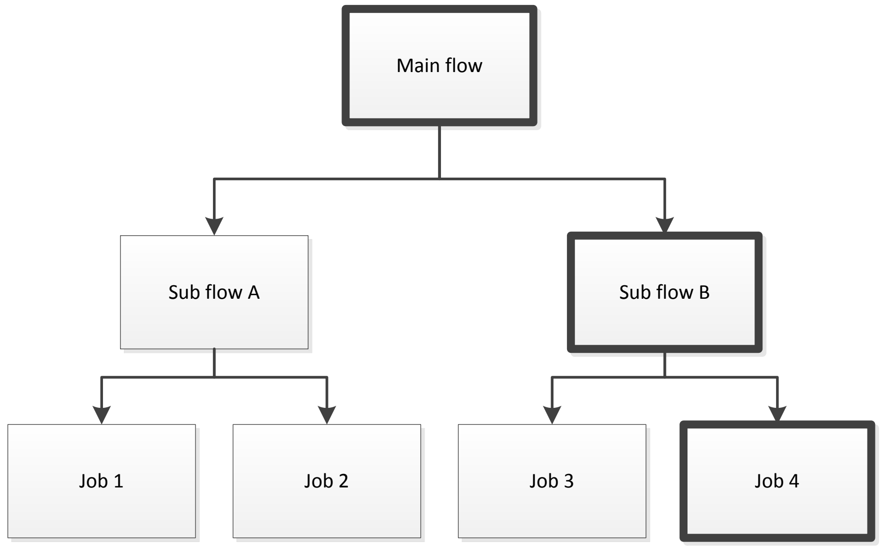


Figure 2 Flow hierarchy

In the first step we export all required information needed to select the flows that need redeployment and their hierarchy. The code selects all JFJob metadata objects and through its associations the deployed jobs and sub flows used in a deployed flow.

Using the same XML map technique as above results in two data sets. One contains all flows and its child objects. Another data set contains only the flows with deployed jobs included in the export. This is achieved by joining the flows on exported deployed jobs.

Next the full hierarchy is composed from the flows and selected flows data set. To loop through the hierarchy efficiently and fast, SAS offers the hash object. The code loops from the bottom flow though all parents until no more parents are found.

```
data flows_full;
 length subfloworjobid flowid flowtreeid $17 flowname $128;
 if _n_ eq 1 then do;
  declare hash h(dataset: "work.allflows");
  h.defineKey( "subfloworjobid");
  h.defineData( "subfloworjobid", "flowid","flowname","flowtreeid");
  h.defineDone();
 end;
 set selectedflows;
 output;
 rc = h.find(key: flowid);
 t = 0;
 do while (rc = 0 and t lt 10);
  t + 1;
```

```
    keep flowid flowname flowtreeid ;
    output;
    rc = h.find(key: flowid);
   end;
  run;
```

METADATA TREE

The export script will refer to each metadata object using its metadata path. In the previous steps each object has a reference to metadata id of the folder it is placed into. This code loops through all the metadata folder objects finding each parent folder until the root of the tree is found, creating a full tree for each folder. The data step hash object is used again to efficiently loop through all the folder objects and generate the full paths.

```
  data full_md_path;
   length folderid $17 uri parent_uri fullpath $500 parent_nm  $200;
   set trees;
   keep folderid fullpath;
   folderid = treeid;
   nobj=metadata_getnobj("omsobj:tree?@id='"||treeid||"'",1,uri);
   rc=metadata_getattr(uri,"Name",fullpath);
   parent_rc=metadata_getnasn(uri,"parenttree",1,parent_uri);
   do while( parent_rc ge 1);
    rc=metadata_getattr(parent_uri,"Name",parent_nm);
    fullpath = catx("/",parent_nm,fullpath);
    uri=parent_uri;
    parent_rc=metadata_getnasn(uri,"parenttree",1,parent_uri);
   end;
   fullpath = "/" || fullpath;
  run;
```

In the end this data set is left joined to each data set with metadata objects resulting in a data set for each object type with the object name, object type and full path. Next all datasets are appended in a specific order to ensure all objects all dependencies can resolve correctly when importing the packages. If the wrong order is used, associations might be lost and jobs can become corrupt.

1. Libraries
2. Tables, dependent on Libraries
3. Transformations
4. External Files
5. Jobs, dependent on Tables, Transformations and External Files
6. Deployed Jobs, dependent on Jobs
7. Deployed Flows, dependent on Deployed Jobs

SHELL SCRIPTS

The final step consists of generating the shell scripts to perform the export and import of packages. To create the packages a string of objects to export is one or more strings like "/metadata/tree/object name (Object Name)".

The length of the string for objects is limited to 1900 characters. If this is too short to contain all objects of a single type, the script will generate statements for multiple packages.

```
data _null_;
 set objects;
 if _n_ = 1 then do;
  put '#!/bin/bash';
 end;
 file "&exportpath./export_&release._&dt.&file_extn." lrecl=25000;
 cmd="/opt/apps/sas/bin/SASPlatformObjectFramework/9.3/ExportPackage -host
&SERVER -port &PORT -user &USER -password &PASS -package
&exportpath./&release._" || strip(put(i,z2.)) || "_" || strip(type) || "_"
|| strip("&dt.") || ".spk -objects " || strip(objects) || " -log
&exportpath./export_&release._" || strip(put(i,z2.)) || "_" || strip(type)
|| "_" || strip("&dt.") || ".log";
 put cmd;
run;

filename chmod pipe "chmod 755
&exportpath./export_&release._&dt.&file_extn.";
data _null_; file chmod; run;
```

Next import scripts are generated with parameters

```
data _null_;
 set objects end=last;
 file "&exportpath./import_&release._&dt.&file_extn." lrecl=25000;
 if _n_ = 1 then do;
  put '#!/bin/bash';
  put 'if [[ -z "$1" || -z "$2" || -z "$3" ]]; then';
  put ' echo "Please enter all required parameters"';
  put " echo ""import_&release._&dt.&file_extn. <servername> <importuser>
<importpassword>""";
  put 'else ';
  put 'timestamp="$(date +%Y%m%dT%H%M%S)"';
  put "sasenv=$(echo $1| cut -d'.' -f 1)";
 end;
 cmd="/opt/apps/sas/bin/SASPlatformObjectFramework/9.3/ImportPackage -host
$1 -port &PORT -user $2 -password $3 -package &exportpath./&release._" ||
strip(put(i,z2.)) || "_" || strip(type) || "_" || strip("&dt.") || ".spk -
target / -preservePaths" || " -log
&exportpath./import_&release._${sasenv}_" || strip(put(i,z2.)) || "_" ||
strip(type) || "_${timestamp}.log";
 put cmd;
 if last then put 'fi';
run;

filename chmod pipe "chmod 755
&exportpath./import_&release._&dt.&file_extn.";
data _null_; file chmod; run;
```

At the end of sprint a master script is created that calls all the individual scripts of user stories ready for promotion from a single command. All the administrator has to do to import a release is run this master script from the shell.

LIMITATIONS

The code contains some limitations. Some object types cannot be changed under change management in Data Integration Studio.

Stored Process

Stored processes generated from DI Studio jobs could be included the same way deployed jobs are included. If a job that has been deployed as a stored process is checked in from a project repository,

Stored processes generated from SAS Enterprise Guide or using hand written code and registered in the SAS Management Console cannot be promoted using the script, because they cannot be checked in from a project repository and cannot be selected based on the change history.

BI Content

We do not promote BI content, like web reports and information maps, from development to production, so no attempt have been made to include these objects into the promotion routing.

Deployment

Since SAS Data Integration Studio 4.7 new batch tools have been added to deploy and redeploy SAS jobs from the command line. Unfortunately these script are written for SAS 9.3 where these commands are not available.

Added the redeployment from the command line to the import scripts is a minor change.

X Server or X Virtual Frame Buffer

To use the batch tools in SAS 9.3 and X Server is required. Because the batch tools only need the X server to be present but have not interaction, the X Virtual Frame Buffer, XVFB, to simulate the existence of an X Server. From SAS 9.4M3 the batch export and import tools can be run with the -disableX11 option avoiding the need of an X Server. The deployment batch tool still requires an X Server in 9.4M3.

UPDATE PHYSICAL TABLES

Deployment of new code can lead to change in table definitions. Many programs add observations to existing table and do not replace the entire table. To execute the code correctly physical data sets must match the metadata definition. In the past each developer was required to write update scripts that were run by the admins after promotion. Sometimes changes were overlooked leading to errors when executing the modified program for the first time.

To reduce the amount of problems during the first run after promotion, we decided to automate the generation of update scripts. After each metadata promotion the physical SAS data sets are compared to the current metadata and scripts are generated to match the physical data set to the metadata. The metadata is always considered the leading definition.

The code compares one library at a time.

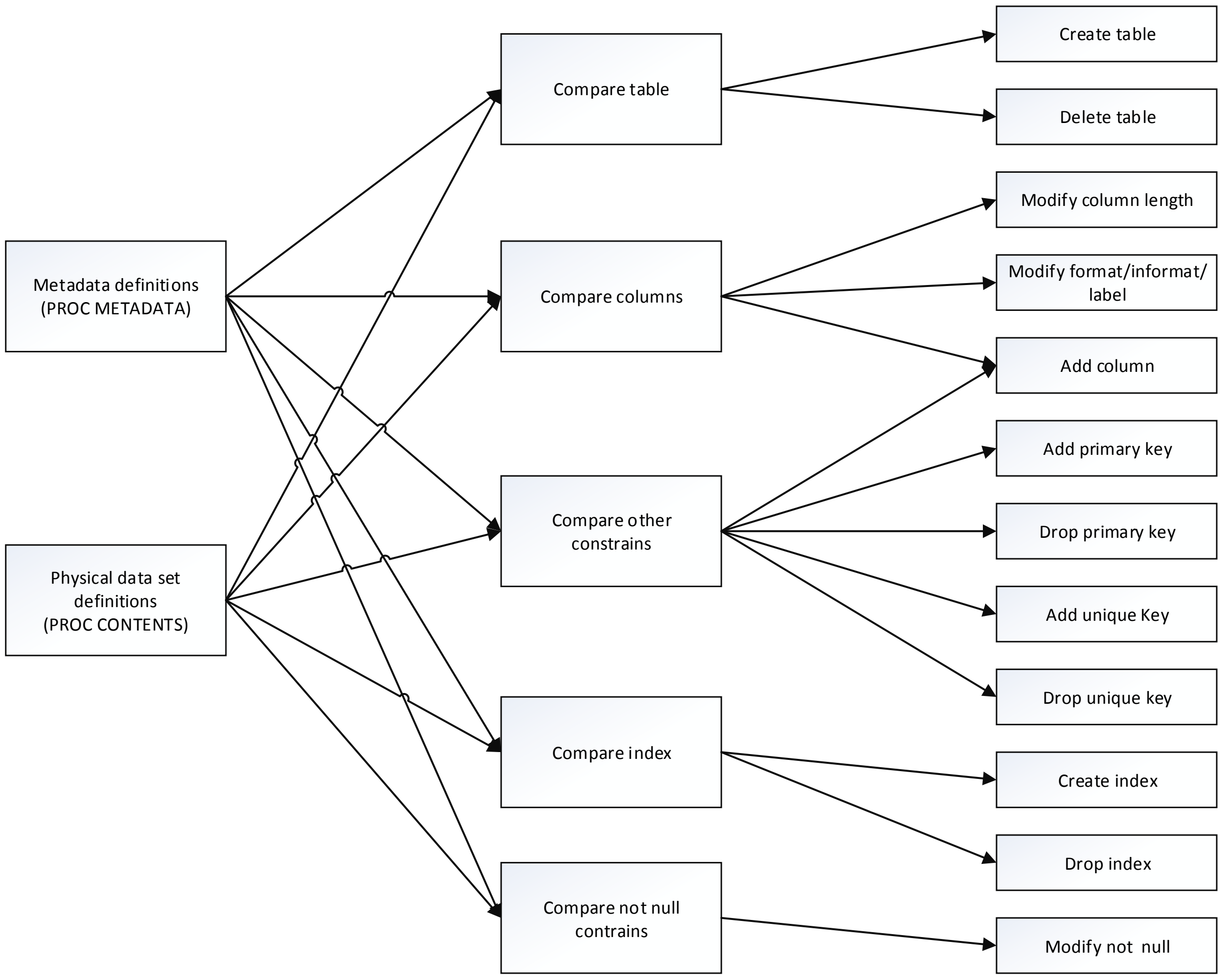


Figure 3 Update Table Generation Process Flow

The basic process flow how to extract and join all information is displayed in figure 3. First we extract the relevant information from metadata and the contents. In the next step both definitions are compared to each other to determine the differences. Finally scripts are generated to update the physical tables.

1. Add tables, columns, keys and constraints that exist in metadata but not as SAS data set
2. Enlarge columns with lengths in metadata larger than in data set
3. Remove tables, keys and constraints that exist as data set but not in metadata
4. Warn for change of column types, deletion of columns and double metadata registrations with different specifications.

We also run the code on development to check if all our tables are in sync with the metadata.

EXTRACT METADATA DEFINITIONS

We extract all table information from the metadata. This can be done using a single metadata query. After extracting all table information the relevant data is joined into workable tables. The script uses the technique also used during generation of the export and import scripts of temporary XML request, XML map and XML response files.

The following objects and attributes are retrieved from the metadata:

Object	Attribute
SAS Library	Id
	Name
	Libref
PhysicalTable	Id
	Name
	MemberType
	SASTableName
	PublicType
Index	Id
	Name
	IndexName
	isClustered
	isHidden
	isNoMiss
	isUnique
UniqueKey	Id
	Name
	isPrimary
Column	Id
	Name
	Desc
	isHidden
	isNullable
	SASColumnLength
	SASColumnName
	SASColumnType
	SASFormat
	SASInformat

The index and keys are split over multiple observations in the dataset. Before those can be compared to the physical table, they have to be reduced to a single observation. A new unique hash is generated for each index by concatenating the names of the columns used for each index, in the order they're used.

We use by processing to concatenate the different observations and output the last observation of each index. We use the same technique for the keys.

```
data tableindexhash;
 set tableindex end=last;
 by libraryref datasetsasname indexname;
 length indexhash $1024;
 retain indexhash;
 if first.indexname then indexhash = upcase(indexsascolumnname);
 else indexhash = strip(indexhash)||' '||strip(upcase(indexsascolumnname));
 if last.indexname then output;
 keep libraryRef datasetId datasetSasName indexId indexName
  IsClustered IsNoMiss IsUnique indexhash;
run;
```

EXTRACT PHYSICAL DEFINITIONS

Physical information from the tables is extracted using the PROC CONTENTS with output written to data sets using ODS OUTPUT. With ODS OUTPUT statement output objects can be written to SAS data sets. The table name and a data set name are specified in the ODS OUTPUT statement. The ODS TRACE statement can help in determine what tables are available within a procedure.

If the library that will be compared is preassigned from the metadata this works out of the box, otherwise the library needs to be assigned first.

```
ods output
 integrityconstraints=vtab_constraint_tmp
 indexes=vtab_index_tmp variables=vtab_columns_tmp;
proc contents memtype=data data=&compareLib.._all_;
run;
ods output close;
```

COMPARE METADATA AND PHYSICAL DEFINITIONS

After both the metadata and physical data set definitions are read into datasets, both sets can be compared. Per type (tables, column, key, index) the new, changed and deleted objects are determined.

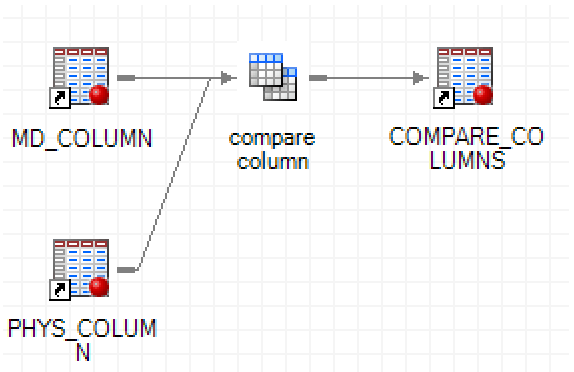


Figure 4 Compare metadata and physical definition for columns

Using a full join the changes are determined and the type of changes are stored in flags in a data set. Only observations with at least one change are kept in the final dataset. The code below is an example to compare the column definitions in metadata to the physical data set.

```
proc sql;
 create table work.compare_columns as
 select
  (coalescec(t1.md_dataset, t2.ph_dataset)) as dataset,
  (coalescec(t1.md_library, t2.ph_library)) as library,
  (case when md_column is missing then 1 else 0 end) as to_delete,
  (case when ph_column is missing then 1 else 0 end) as to_add,
  (case when ph_datatype ne md_datatype then 1 else 0 end) as ne_datatype,
  (case when ph_label ne md_label then 1 else 0 end) as ne_label,
  (case when ph_length ne md_length then 1 else 0 end) as ne_length,
  (case when ph_format ne md_format then 1 else 0 end) as ne_format,
  (case when ph_informat ne md_informat then 1 else 0 end) as ne_informat,
  (case when ph_length gt md_length then 1 else 0 end) as ph_len_gt_md_len,
  t2.ph_library, t2.ph_dataset, t2.ph_column, t2.ph_order, t2.ph_length,
  t2.ph_datatype, t2.ph_label, t2.ph_format, t2.ph_informat,
  t1.md_library, t1.md_dataset, t1.md_column, t1.md_order, t1.md_length,
  t1.md_datatype, t1.md_label, t1.md_format, t1.md_informat
 from work.md_column t1
  full join work.phys_column t2
   on (t1.md_library = t2.ph_library) and (t1.md_dataset = t2.ph_dataset)
and (t1.md_column = t2.ph_column)
 where (max(calculated todelete, calculated toadd, calculated ne_datatype,
calculated ne_label, calculated ne_length, calculated ne_format, calculated
ne_informat)) >= 1;
quit;
```

CREATE ALTER TABLE SCRIPTS

The final step is generating SAS scripts that will alter the physical table to correspond with the definitions in the metadata. As shown in figure 3, 12 separate SAS scripts are generated.

```
data _null_;
 file "&pad./06_modify_column_format_informat_label_&compareLib..sas"
lrecl=1024;
 set selection_columns_modify (where=(todelete=0 and toadd=0 and
     ne_datatype=0 and (ne_format=1 or ne_informat=1 or ne_label=1)))
     end=last;
 by dataset;
 if _n_ = 1 then put 'proc datasets library=' library 'nolist;';
 if first.dataset then put " modify " dataset ";";
 if ne_format = 1 then put '  format ' md_column md_format +(-1)';';
 if ne_informat = 1 then put '  informat ' md_column md_informat +(-1)';';
 if ne_label = 1 then put '  label ' md_column "='" md_label +(-1)"';";
 if last then put 'quit;';
run;
```

The scripts are numbered and should be run in ascending order of the numbers, because of dependencies between the steps. An index for example can only be created after the old definition is deleted and required columns have been added to a data set. The dependencies lead to the following order:

- 01_create_table.sas
- 02_drop_primary_key.sas
- 03_drop_unique_key.sas
- 04_drop_index.sas
- 05_modify_column_length.sas
- 06_modify_column_format_informat_label.sas
- 07_add_column.sas
- 08_modify_null_constraint.sas
- 09_add_primary_key.sas

- 10_add_unique_key.sas
- 11_index_create.sas
- 12_drop_table.sas

CONCLUSION

After development each developer creates a package for the user story he has been working on. The developer is responsible for deployment of the package to the test environment. This serves as check that everything needed for a release is included in the package and that the newly imported jobs can be deployed correctly. By automating we reduced the time needed to create packages and the chance of errors caused by forgetting to include required objects.

The total process reduced to prepare for and execute a release from days to hours while reducing the number of errors encountered during deployment. The scripts to update the physical tables reduced the number of errors encountered when running new code for the first time.

The automation of promotion has been a critical success factor for the team to increase new features and functionality developed and released each sprint. Not only do developers benefit from the automation, administrators also experience less pressure from releases on their tasks. The deployment is faster and emergency fixes to make a release working are less frequent.

ACKNOWLEDGMENTS

The authors would like to thank John de Kroon, Stephan Minnaert, Yat Pang, Rolf Tijssens and Arnoud Weber for the feedback given and improvements made to the code while using the automated deployment process.

RECOMMENDED READING

- Oltsik, Myra A. (2008) *"ODS and Output Data Sets: What You Need to Know"*. Proceedings of the SAS Global Forum 2008, Cary, NC. SAS Institute, Inc.
- *SAS® 9.3 Open Metadata Interface: Reference and Usage*
- *SAS® 9.3 Metadata Model: Reference*
- *SAS® 9.3 Intelligence Platform: System Administration Guide, Second Edition*
- *SAS® Data Integration Studio 4.4: User's Guide*
- *ODS and Output Data Sets: What You Need to Know*

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Laurent de Walick
PW Consulting
laurent.de.walick@pwconsulting.nl
http://www.pwconsulting.nl

Bas Marsman
NN Bank
bas.marsman@nn.nl
http://www.nn.nl