Paper 102-2009

Parameter Driven Data Integration Using SAS® Stored Processes

Evangeline Collado, University of Central Florida, Orlando, FL

ABSTRACT

The data warehouse at this large, metropolitan, state university consists of term and annual census data. Loading data into the warehouse is a data integration (DI) process that cannot be scheduled to run at a particular time each day as it requires human intervention to specify the appropriate time period for the new data. The data transforms in the process flow could be changed each term or year, but each node that requires this term parameter would need to be changed. We desired a DI process that would prompt the data integration developer to provide this input when the job is executed and it would remain constant throughout the process flow. This paper details one particular DI process that was developed and deployed as a SAS stored process using SAS® Data Integration Studio 3.4, which offers the developer the ability to create parameters, and executed using SAS® Enterprise Guide 4.1, which prompts the user to supply the parameter. The intended audience is anyone who faces similar data integration challenges and requires this type of solution.

WHO WE ARE

The mission of the Office of Institutional Research (IR) is to provide information of the highest quality, which is both timely and easily accessible, and to facilitate and enhance decision-making, strategic planning, and assessment at the University of Central Florida. The scope of the unit's responsibility embodies the university, state and federal agencies, and the general public. One of the primary tasks of this office was to develop, implement, and maintain an enterprise-wide data warehouse that would be the source for all official statistical analyses and reporting of university information. Our office chose SAS® Enterprise Data Integration (DI) Server and SAS® Enterprise Business Intelligence (BI) Server to complete this task.

INTRODUCTION

One of the many reporting applications developed by the IR office several years ago provides information on student credit hours (SCH) and full-time equivalents (FTE). This application was originally developed using SAS® Base software and surfaced to the UCF community and general public via the Web using SAS® IntrNet software. The data that drives this application was extracted from our student census files that were stored in Microsoft Access tables. The application provides the information in a drill-down manner using dynamic links that are generated in each successive program.

As part of the conversion process to add improved functionality to the application using the new BI tools provided in the SAS Enterprise BI Server software suite, it was decided that, because the data are multidimensional, an OLAP (OnLine Analytical Processing) cube would be the best solution. Since the census data are now loaded into the data warehouse, we could design a DI process that would extract the data from the dimension tables and load all the necessary reporting fields into a fact table (SCH_FACT). An OLAP cube could then be developed using SAS® OLAP Cube Studio software and an information map for use with other SCH reporting needs could be developed using SAS® Information Map Studio software. With this functionality, all the necessary historical data could be loaded in a single extract from the warehouse dimension and lookup tables (BASE engine). Additional information could be extracted from the University's Enterprise Resource Planning (ERP), or Student Information, transactional system (ORACLE engine). However, when data from a new term is to be added to the SCH_FACT table, the DI process would need to be modified manually to supply the time period for the data to be extracted. A re-usable, easily maintained process would need to be developed.

DATA INTEGRATION DEVELOPMENT

Before the process flow could be created, a thorough examination of the current application was required, as all necessary reporting fields and their sources had to be identified. The next step was to create a data model or field mapping diagram that described the target table (SCH_FACT) and each source table's contributing fields. For data elements that would be built using expressions, the logic was clearly detailed in the document. This preliminary work made the development of the data integration process flow shown in Figure 1 more straightforward.

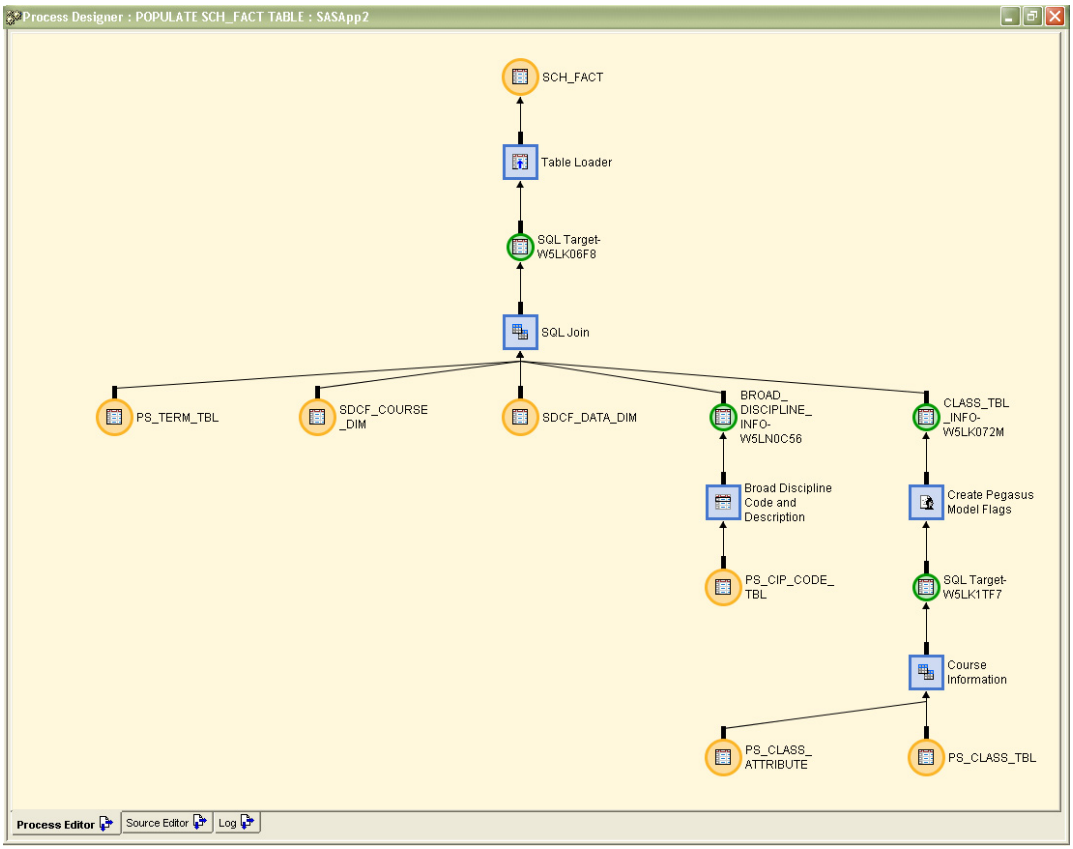
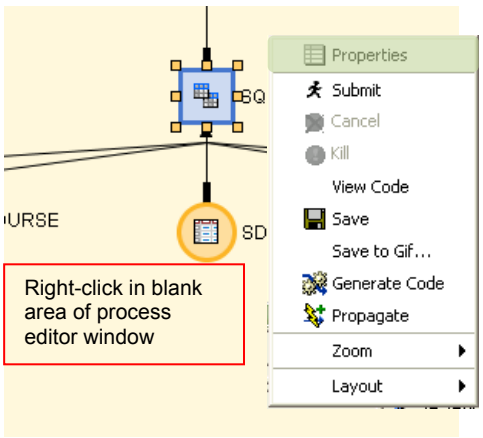


Figure 1. Data Integration Process Flow to Populate SCH_FACT Table

The Course Information and other SQL Join nodes in this process are where we needed to restrict the data selected to a particular term. The appropriate tables are joined on the term field such that the data returned is what meets that criterion. Other DI processes that we have developed have multiple nodes that filter on term. We do not want to have to look at each node every time we run the process to see what transforms need modification. We had experience using SAS Base software to define a local or global macro variable in the program that would supply the value needed in subsequent code but SAS DI Studio generates the code for the SQL join data transform. How could we define a global macro variable that the entire process would use? The *SAS® Data Integration Studio 3.4 User's Guide* provides information about parameterized jobs and iterative jobs but that did not provide the required solution. What we needed was an input parameter and a way to prompt for the value at run time. Further searching of the guide produced a section titled "(Optional) Parameters for User Input" that details how a Web client user could supply a value for a job that has been deployed for execution by a Web service client. Another section of the guide details how to deploy a job as a SAS stored process - "a SAS program that is stored on a server and can be executed as required by requesting applications" (SAS Institute Inc. 2007). Parameters can be defined in SAS Enterprise Guide to prompt you for input, and you can run a stored process in this client tool, so this seemed like a viable solution to our challenge. We just had to figure out how the code could be generated with the reference to the macro variable.



To create a global parameter for the entire job you right-click in the blank area of the process editor and select **Properties** (Figure 2). Select the **Parameters** tab in the Properties window as shown in Figure 3. You can create, edit, delete, and import parameters here. The STRM parameter that we created for the term identifier is shown in Figure 4. The data integration process would need to be tested before it could be deployed as a stored process with parameters; thus, development would begin with a single term as the default value of the parameter and then changed to %SUPERQ(STRM) when we knew the process executed correctly.

Figure 2. Right-Click Menu

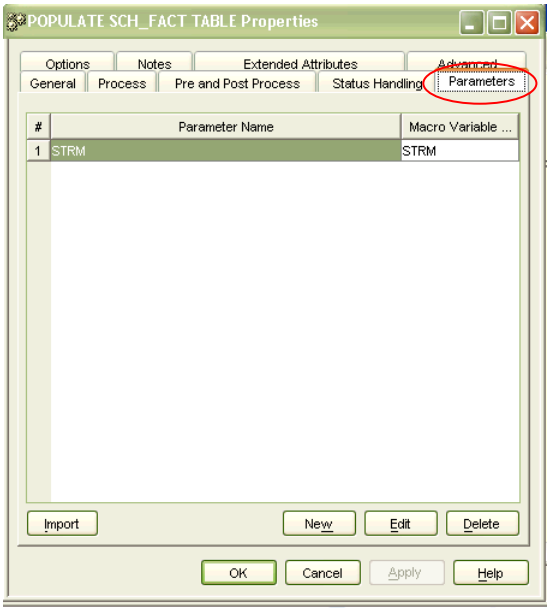


Figure 3. Process Editor Properties Window

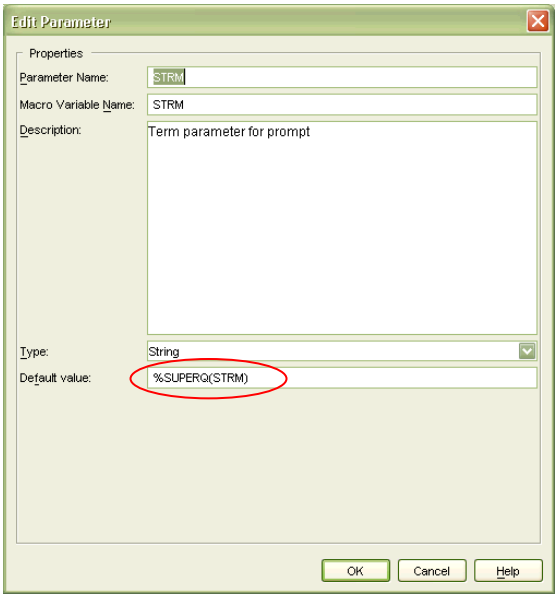


Figure 4. Parameter Settings

The code that is generated is written such that this parameter will receive the value prompted for in the stored process (Figure 5). We tried omitting the default value but the code was generated with a null value so the process did not execute correctly. By using parameters we are able to use the macro variable in each where clause of the SQL Join nodes.

```
/*************************************************************
 * Job:           POPULATE SCH_FACT TABLE                    A
 * Description:   Create an SCH_FACT table containing data fr
 *                 SDCF_COURSE_DIM and other tables for SCH r
 *
 * Repository:    Foundation                                 A
 * Server:        SASApp2                                    A
 *
 * Source Tables: SDCF_COURSE_DIM -                          A
 *                SDCF_DATA_DIM -                            A
 *                PS_TERM_TBL -                              A
 *                PS_CIP_CODE_TBL -                          A
 *                PS_CLASS_TBL -                             A
 *                PS_CLASS_ATTRIBUTE -                       A
 *
 * Target Table:  SCH_FACT - SCH.SCH_FACT                    A
 *
 * Generated on:  Thursday, February 5, 2009 8:18:48 AM EST
 * Generated by:
 * Version:       9.1.20060626.51875
 *************************************************************

/* Parameter default value(s) for POPULATE SCH_FACT TABLE  */
%let STRM = %SUPERQ(STRM);
```

Process Editor | Source Editor | Log

Figure 5. Generated Source Code

The Operand in Figure 6 and Figure 7 was set to the quoted macro variable name, “&STRM”, and the where clause that was generated also contains this value.

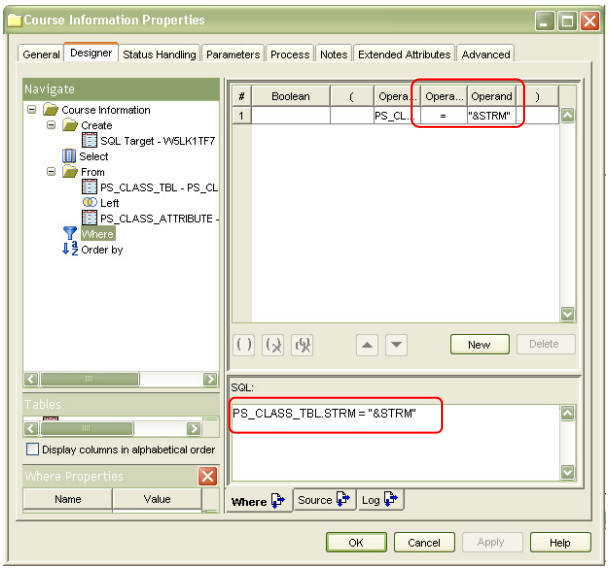


Figure 6. Course Information Where Clause

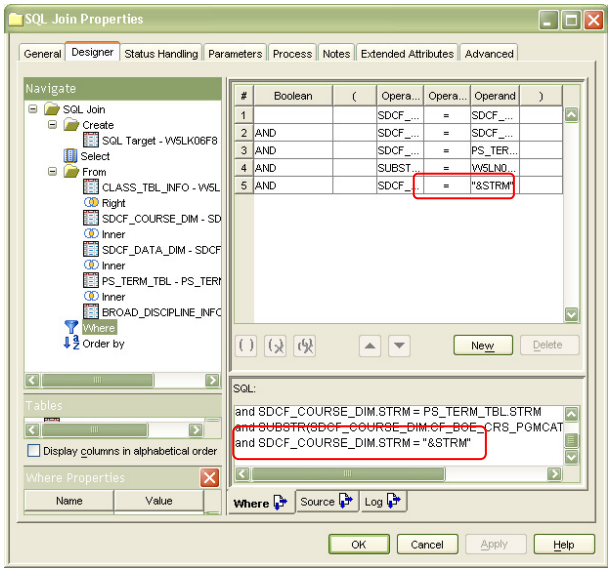


Figure 7. SQL Join Where Clause

The next step was to deploy the job as a stored process. To do this you right-click on the job and select **Stored Process → New…** (Figure 8). The Stored Process Wizard will guide you step-by-step to complete the properties for the new stored process. Figure 9 shows the properties of the stored process we deployed from the POPULATE SCH_FACT TABLE job. The name and description in the **General** tab are entered for you by the application when you start the wizard.

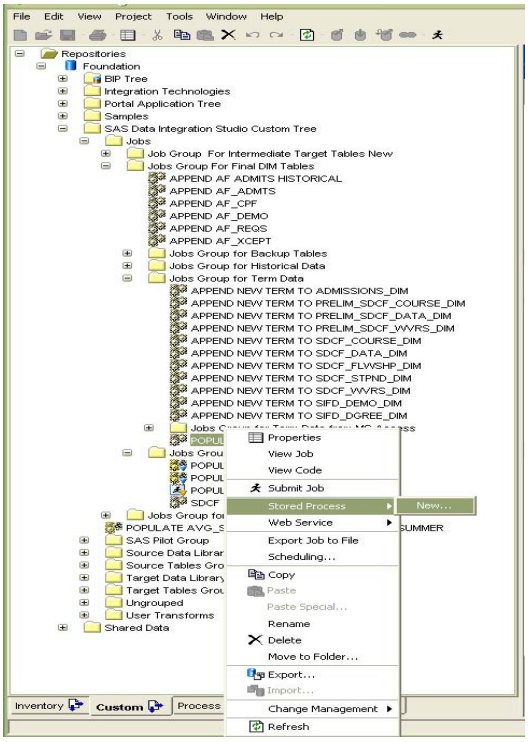


Figure 8. Deploy Job as a Stored Process

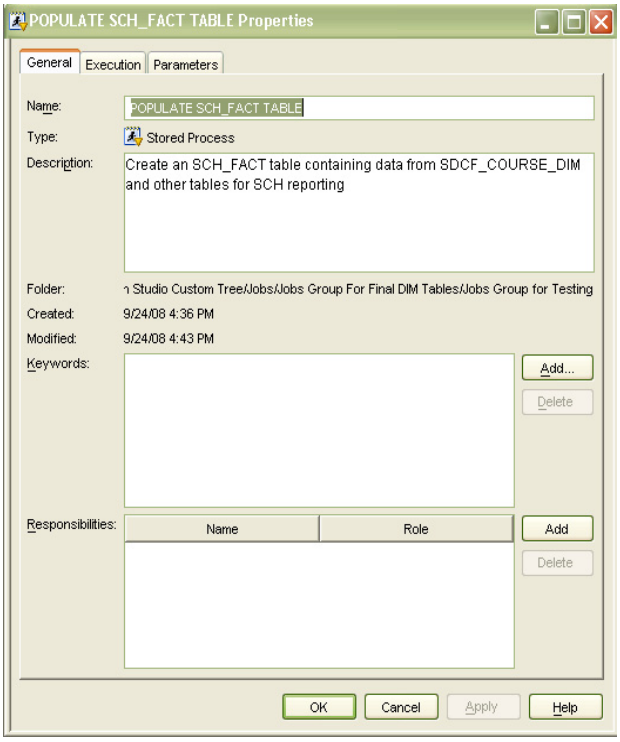


Figure 9. General Properties

The **Execution** properties specify the SAS server for the stored process execution, the path for the program file that will be generated, the name for this generated file, input and/or output properties (Figure 10). If the desired path has not been previously defined in your metadata then you can click on the **Manage…** tab to add a new path (source code repository). The **Parameters** tab as shown in Figure 11 is where we added the global parameter we specified during the parameter creation process. The next step is to execute the stored process in SAS Enterprise Guide.

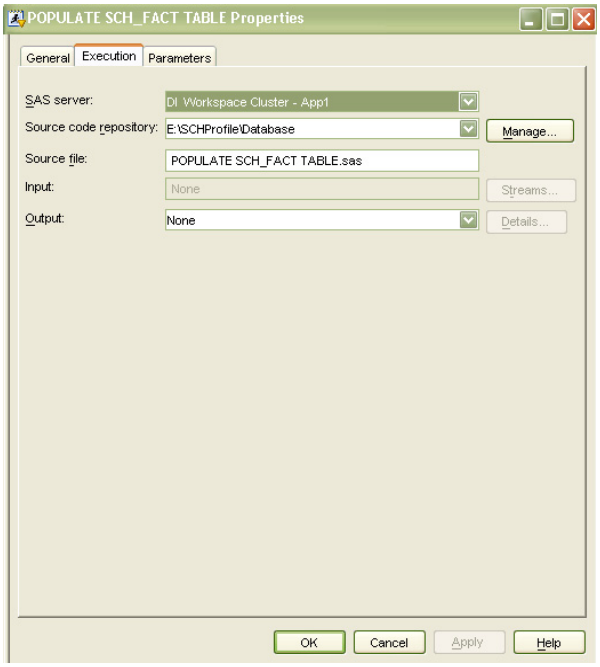


Figure 10. Execution Properties

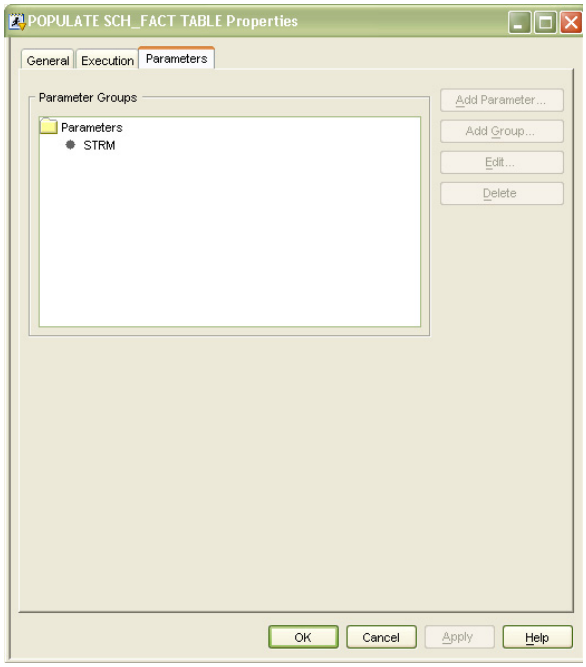


Figure 11. Parameters Properties

If changes are made to the DI job, or the job is promoted from one environment to another, then the stored process must be redeployed. Selecting **Tools → Redeploy Jobs to Stored Processes** from the **Menu Bar** will redeploy **all** jobs so you need to right-click on the job, select **Stored Process → *Name of stored process* → Redeploy.** The user's guide explains the deployment steps in greater detail.

STORED PROCESS EXECUTION

The stored process that is deployed from a SAS Data Integration Studio job can be executed from many SAS Applications such as SAS Enterprise Guide, SAS® Add-In for Microsoft Office, SAS® Information Delivery Portal, and others as detailed in the *SAS® Data Integration Studio 3.4: User's Guide*. We chose to use SAS Enterprise Guide for this project. To begin, open SAS Enterprise Guide and begin a new project. Select **File → Open → Stored Process** from the **Menu Bar.** Navigate the SAS Data Integration Custom Tree to select the stored process from the appropriate folder (navigation differs depending on version of client software). Select the stored process in the Process Flow and right-click to open the Stored Process Manager. **General Information** allows you to specify a name, description, and searchable keywords for the stored process. The **SAS Code** window (Figure 12) should show the generated code from the data integration job. We needed to select all options available in the **Include code for** drop-down list to resolve the macro variable. The **Metadata Location** window is where you specify the location of the metadata for the stored process. If the location you prefer is not available from the **Choose location...** button then you will need to have a metadata location created in SAS® Management Console. The **Execution Environment** window provides the option to change the properties you specified when you deployed the job as a stored process.

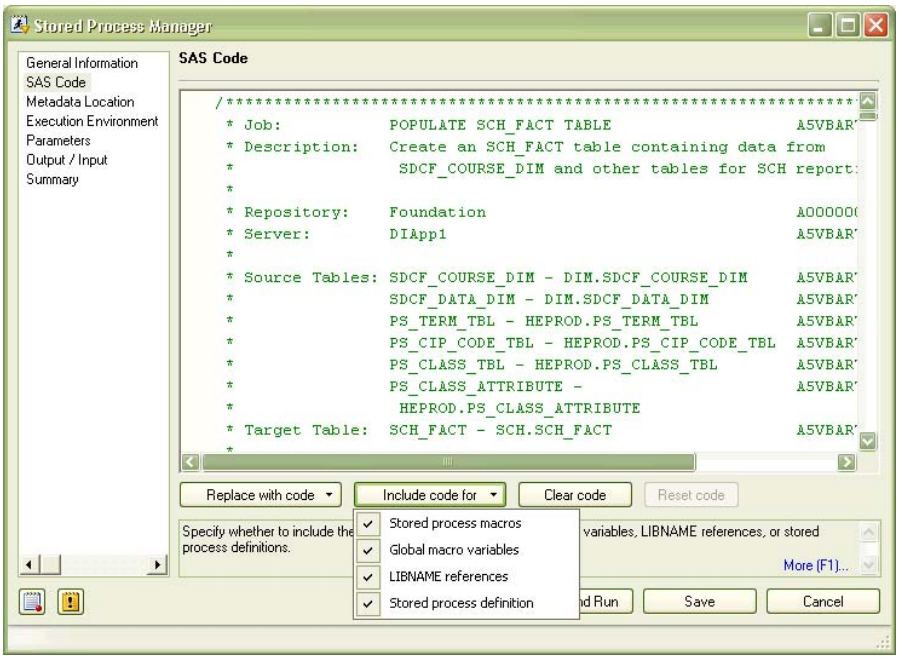


Figure 12. SAS Code Window and Options

The **Parameters** window, as shown in Figure 13, is where you can add, edit, or delete any parameters for the stored process. We selected **Add** → **Parameters from SAS Code** to add the STRM parameter to this stored process. Other macro variables from the SAS code were skipped.

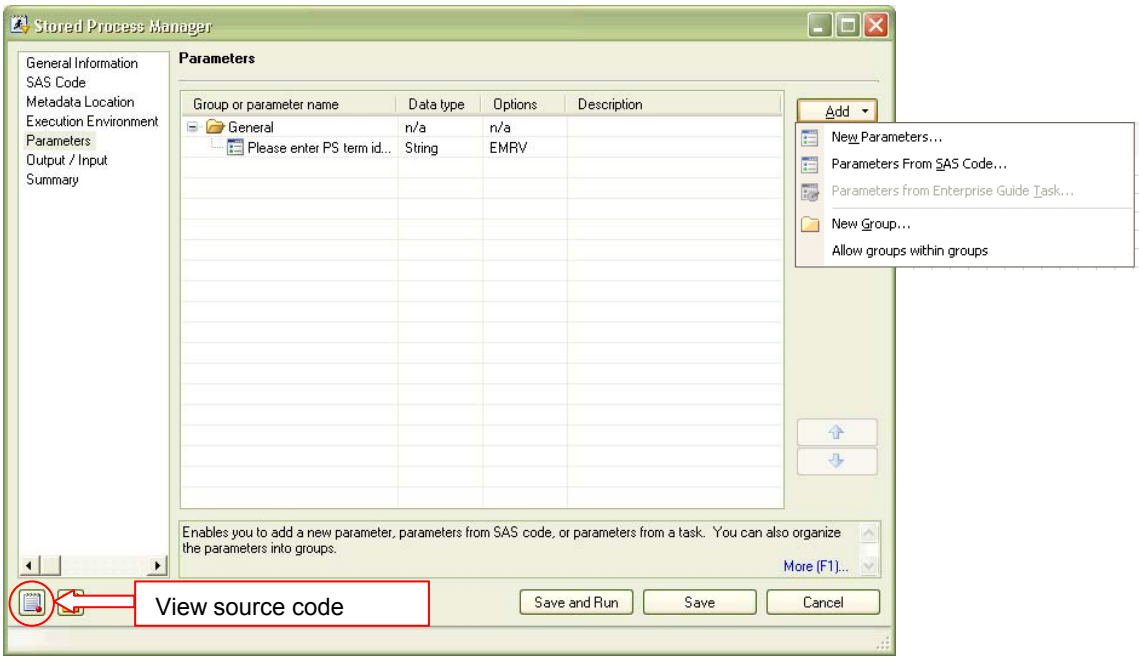


Figure 13. Stored Process Parameter Window in SAS Enterprise Guide

Output/Input options were set to **None** in this project. The **Summary** window provides summarized details about the current stored process. The editor icon allows you to view the generated source code for the stored process. Figure 14 shows the parameter code that Enterprise Guide added. This code is needed so the application will prompt you for the specified parameter(s) at run time.

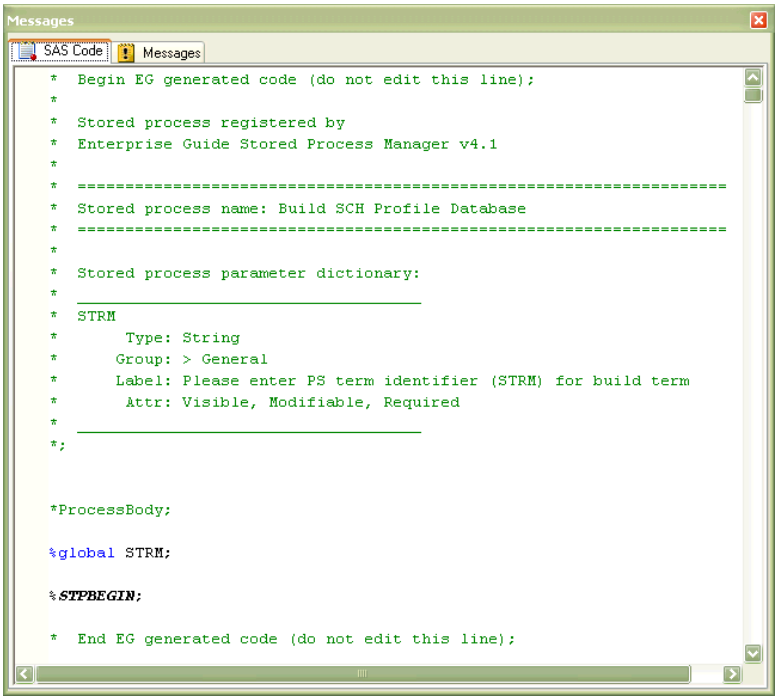


Figure 14. Stored Process SAS Code Window

To execute the stored process you can select **Save and Run** in the Stored Process Manager or you can right-click on the node in the Process Flow and select **Run *Name of Stored Process*** as shown in Figure 15. When we execute the stored process for the SCH_FACT table append, the parameter prompt window (Figure 16) requests the value for the term to be appended (STRM). If the process does not execute properly there will be a red X on the node and the errors will appear in the log.

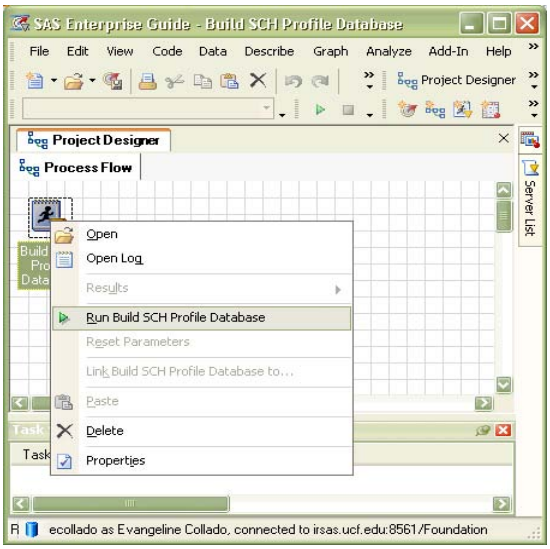


Figure 15. Run Stored Process

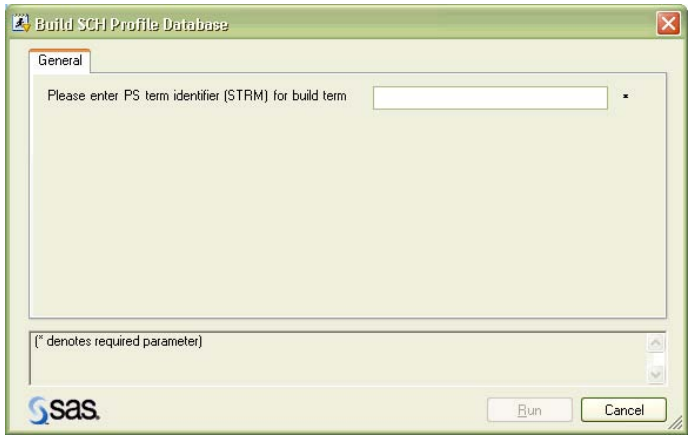


Figure 16. Prompt for Parameter Value

Our process ran successfully, after much trial and error, and the data for the most recent term was added to the table. We have a reusable, easily-maintained process that can be executed by any member of our DI development team.

CONCLUSION

Due to the time period requirements for loading new data into the enterprise data warehouse a more robust DI process is required at the University of Central Florida. The SAS® 9.1.3 Intelligence Platform provides multi-faceted solutions to fulfill requirements in many environments. The capability to create parameters in a data integration process using SAS Data Integration Studio 3.4 was the stepping stone needed to accomplish our task. We identified that the tool would enable a developer to use parameters in an iterative job but we still needed a way to prompt the user for those input parameters when the process was executed. Discovering that the job can be deployed as a stored process, and having experience developing stored processes that prompt for user input, led to the customized solution we needed to work through this data integration challenge. Now it has become a staple in our data integration repertoire.

REFERENCES

SAS Institute Inc. 2007. *SAS® Data Integration Studio 3.4: User's Guide*. Cary, NC: SAS Institute Inc. Available at http://support.sas.com/documentation/onlinedoc/etls/usage34.pdf.

CONTACT INFORMATION

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

Name: Evangeline (Angel) Collado
Enterprise: University of Central Florida, Office of Institutional Research
Address: P.O. Box 160021
City, State ZIP: Orlando, FL 32816-0021
Work Phone: (407) 823-4968
Fax: (407) 823-4769
E-mail: ecollado@mail.ucf.edu
Web: www.iroffice.ucf.edu