Paper PA-212

Your Local Fire Engine Has an Apparatus Inventory Sheet and So Should Your Software: Automatically Generating Software Use and Reuse Libraries and Catalogs from Standardized SAS® Code

Troy Martin Hughes

ABSTRACT

Fire and rescue services are required to maintain inventory sheets that describe the specific tools, devices, and other equipment located on each emergency vehicle. From the location of fire extinguishers to the make, model, and location of power tools, inventory sheets ensure that firefighters and rescue personnel know exactly where to find equipment during an emergency, when restocking an apparatus, or when auditing an apparatus' inventory. At the department level, inventory sheets can also facilitate immediate identification of equipment in the event of a product recall or the need to upgrade equipment. Software should be similarly monitored within a production environment, first and foremost to describe and organize code modules—often SAS® macros—so they can be discovered and located when needed. When code is reused throughout an organization, a reuse library and reuse catalog should be established that demonstrate where reuse occurs and to ensure that only the most recent, tested, validated version of code modules are reused. This text introduces SAS software that automatically parses a directory structure, parses all SAS program files therein (including SAS programs and SAS Enterprise Guide project files), and automatically builds reuse catalogs from standardized comments within code. Reuse libraries and reuse catalogs not only encourage code reuse but also facilitate backward compatibility when modules must be modified because all implementations of specific modules are identified and tracked.

INTRODUCTION

Software *reusability* is defined as the "degree to which an asset can be used in more than one system, or in building other assets."[i] In this text, *assets* represent only software programs or modules (such as SAS macros) that can be reused in subsequent software, but in a broader sense, *assets* can also refer to software documentation, risk templates, quality controls, data dictionaries, and a host of other software-related artifacts. Two knowledge management artifacts commonly implemented to document and manage assets and to facilitate reusability include the reuse library and the reuse catalog. Together, these tools facilitate documentation, organization, search, and retrieval of software so that developers can locate, understand, and implement code more effectively and efficiently. In many environments, these artifacts additionally include information about software risk so that SAS practitioners can understand the performance and relative quality of modules as well as the intended usage. Armed with this information, SAS practitioners can better assess whether code can be reused in its entirety, repurposed to meet additional business needs, or redesigned and developed from scratch.

Software reusability is thwarted when software is not adequately documented and organized, and unfortunately common in many software development environments, documenting code may be considered an afterthought rather than an integral element of development. In essence, software documentation is sometimes conceptualized as a distinct phase of the software development life cycle (SDLC) that follows software release and which overlaps with software operations and maintenance (O&M) activities. To the contrary, a widely held software development best practice is to document software—however lean or fat that documentation may be—into the software development phase, and possibly as early as software design. In doing so, whether operating in Waterfall or Agile development environments, this ensures that necessary documentation will be appropriately prioritized and provided sufficient resources (i.e., personnel) to be successful.

The automation of documentation can be a tremendous benefit to software development because developers can produce useful documents with little effort and, as demonstrated in this text, in some cases only through comments maintained within software. By standardizing the type and content of comments that appear in SAS code, software can be parsed automatically, all comments extracted and organized, and successful external documentation created through SAS reporting functionality. In addition to parsing comments, SAS functionality can also be parsed. For example, the %MACRO statement denotes the definition of a SAS macro, so the definition of all SAS macros can be

collected, parsed, and organized with ease. Parsing of the %INCLUDE statement can similarly demonstrate dependencies that software may have on external programs and macros.

While numerous SAS white papers demonstrate both the successful standardization and parsing of comments within SAS program files to produce documentation automatically, a common obstacle within environments using SAS Enterprise Guide has been the disparity in file formats. SAS program files (e.g., those with the .SAS extension that are created by the SAS Display Manager) are ASCII text files that can be readily ingested into SAS data sets for manipulation and analysis, whereas the SAS Enterprise Guide files (e.g., project files with the .EGP extension) are compressed, zipped files that must be extracted and interrogated before internal code can be extracted and analyzed. With the introduction of the ZIP access method within the FILENAME statement in SAS 9.4, this hurdle has been overcome, and SAS software can now read both program and project files (i.e., .SAS and .EGP files) to support automated document generation and other analytic endeavors.

This text introduces the SCAVENGER macro suite that iteratively generates a list of all SAS program and project files within specified folders, extracts SAS programs from all project files, and parses all programs to produce reporting automatically that supports software documentation, organization, and search. Once SCAVENGER has aggregated the collective metadata describing a SAS environment, a straightforward REPORT procedure demonstrates the conceptual creation of a reuse catalog. With this pain-free documenting in place, SAS environments can dramatically improve their software reusability and reuse posture, further driving more efficient software development. And, because SCAVENGER itself is coded through modular software design that facilitates flexibility and reuse, other potential uses of SCAVENGER are also discussed. Scavenger must be run on SAS 9.4 or higher in a Windows environment but offers out-of-the-box functionality that can interrogate thousands of SAS projects and programs!

REUSE LIBRARY

A *reuse library* is defined as "a classified collection of assets that allows searching, browsing, and extracting."[ii] Within this text, because *asset* references only software, a reuse library is nothing more than an organized software repository. Thus, reuse libraries can comprise a simple Windows directory structure on a shared network, implementation of the SAS Autocall Macro Facility, a SharePoint site, even more complex knowledge management software that provides software versioning, or any environment or tool along this continuum. To be effective, however, developers must be able to access a reuse library efficiently to research what software solutions have already been developed. When reuse libraries are poorly maintained, SAS practitioners are more likely to maintain individual rather than shared code bases because they cannot successfully locate software when needed.

In addition to promoting efficiency and effectiveness, reuse libraries also must be secure. Whenever developers are forced to check in their code to a single, shared repository, the chance for corruption or overwriting increases so SAS practitioners accustomed to developing in stovepipes will naturally be wary of ceding some control over their SAS babies. To facilitate security of and faith in reuse libraries, shared code repositories typically espouse security measures such as version control or backups to guard against unwanted or untoward software modifications, and possibly user auditing and permissions to guard against unintended or malicious modifications. Only with shared code repositories can developers be sure they are using (or modifying) the most accurate, current, or complete version of specific software. Moreover, this level of coordination can make the software development process much more efficient, as a single code base can be tested, validated, and approved rather than wasting effort on testing various versions of similar software.

Reuse libraries are a critical first step toward organization of a collective software base for an environment, enabling basic search functionality to locate software modules. However, through the synthesis and analysis of collective software into a refined artifact, additional information and utility can be gained. Thus, reuse catalogs can be conceptualized as the metadata repositories that describe reuse libraries. For example, to determine the number of times and ways in which a specific SAS macro is reused throughout an organization, a developer could conduct a global search of his network to produce a disorganized, dizzying array of information. Where reuse catalogs are incorporated, however, the developer would be able to access this information directly through a standardized structure. Other benefits can include summary metadata such as cryptographic checksums of program files (to demonstrate software integrity), line counts, or versioning information that can be calculated automatically and added to reuse catalogs to promote greater understanding of and security in software modules.

REUSE CATALOG

A *reuse catalog* is defined as "a set of descriptions of assets with a reference or pointer to where the assets are actually."[iii] Reuse catalogs rely on the underlying structure and content of reuse libraries but facilitate exploration and retrieval through additional metadata and information. Typical information found within a reuse catalog can include:

- software path and file name
- program name (if a program is imbedded within a SAS Enterprise Guide project)
- software description
- link to program files
- creation time-date stamp
- last update time-date stamp
- additional versioning information
- file size
- lines of code
- cryptographic checksum (used to validate program file integrity)
- risk (including known threats, software vulnerabilities, and technical debt)
- program prerequisites or dependencies (including other software that use/reuse the program)
- program inputs or outputs

The SCAVENGER program is capable of retrieving and parsing most of these metadata and generates a denormalized data set with unique observations demonstrating individual SAS programs. Thus, when a SAS program file is encountered, a single observation is generated. Conversely, when a SAS project file is encountered, this generates one entry for each program (linked or imbedded) found within the project. To help elucidate program prerequisites and dependencies, all macro definitions and referenced program files (via the %INCLUDE statement) are summarized for each program. The creation of user-defined global macro variables (via the %GLOBAL statement) is also summarized to facilitate deconfliction between SAS modules and macros (since identically named global macro variables can unintentionally overwrite each other). SCAVENGER benefits SAS practitioners who would otherwise be forced to sift through code manually and who would be less likely to reuse existing code.

Moreover, because SCAVENGER iteratively parses all code within an infrastructure, subsequent reuses of software are cataloged. For example, the FINDVARS macro might be created and used to generate a space-delimited list of all variables found within a data set. Because of its flexibility and generalizability, FINDVARS would be an ideal candidate for inclusion in a reuse library, at which point other developers could begin to use it in their respective software products. However, if FINDVARS needed to be modified—to increase functionality, improve performance, or reduce vulnerabilities—developers would have to check all individual uses of FINDVARS to ensure the proposed modifications were backward compatible to current usage and did not break software products already using it. Thus, reuse is facilitated by reuse catalogs that depict all software using specific software modules.

Another tremendous advantage of reuse catalogs is their ability to drive the reuse-versus-redevelop decision, in which developers are faced with the decision of whether to use (or cannibalize) existing code to build a software product or to redesign and redevelop it from scratch. When inline comments within SAS programs include information about software capabilities, use cases, best practices, and vulnerabilities, developers can make more informed decisions about whether, how, and to what degree to reuse existing software modules in future software products. When standardized, these metadata (included in SAS inline comments) can be automatically parsed and extracted.

STRUCTURE OF SAS ENTERPRISE GUIDE PROJECT FILES

Because the parsing and analysis of Enterprise Guide project files has represented an obstacle to SAS practitioners in the past, some background on the structure of project files is warranted and will facilitate the reuse and generalizability of the READXML and READCODE macros (included in Appendix A) that read and parse the project file structure and imbedded programs, respectively. Figure 1 demonstrates the typical "Process Flow" view of a sample SAS Enterprise Guide project file. In this example, the Initialize program executes first, after which branches bifurcate to two programs, Macros and Create_data. The project note describes this prerequisite.

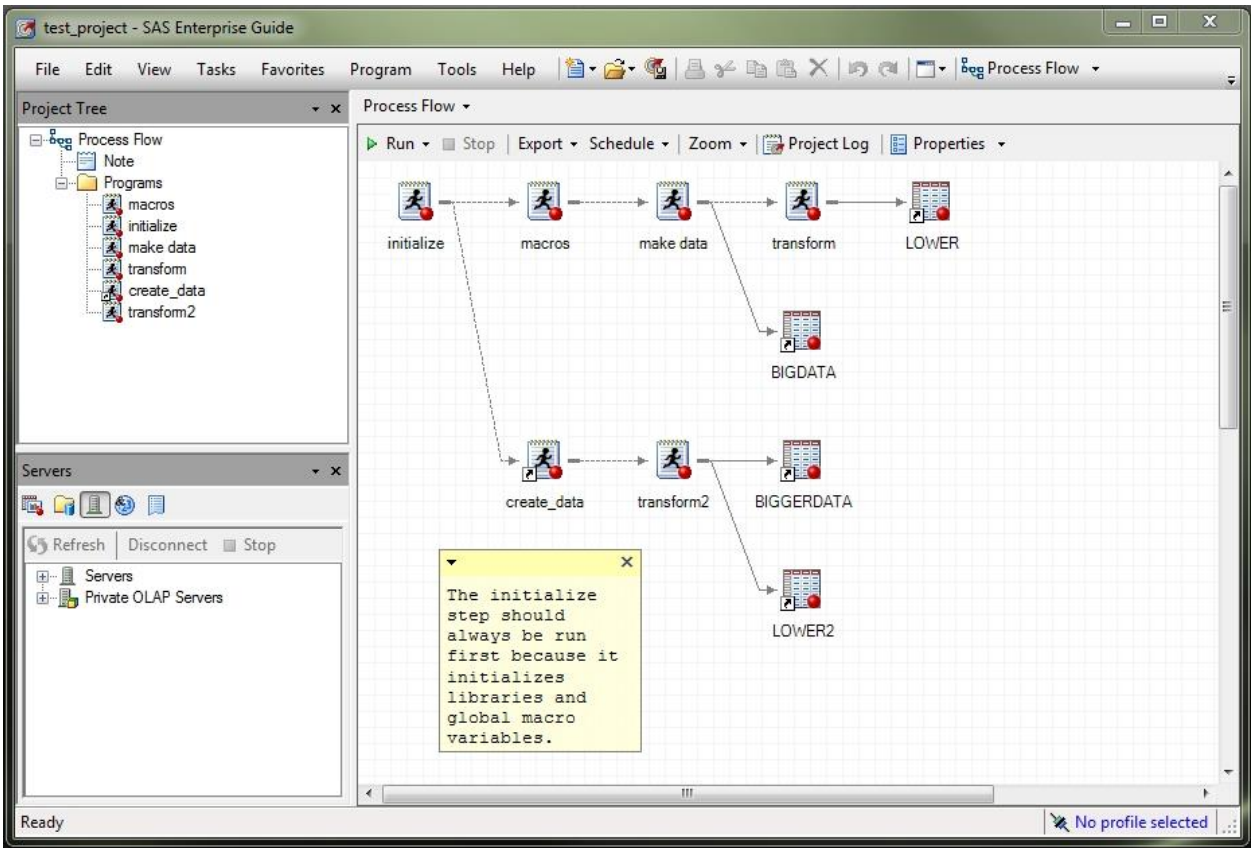

Figure 1. Process Flow View of a SAS Enterprise Guide Project File

The Initialize and Macros programs are both imbedded programs, in that they exist only inside the project file itself. A program can be imbedded by right clicking the workspace and selecting "New Program." However, because imbedded programs are saved in the project file, they are not available for external use or application. The Create_data program conversely represents a linked program, in which the code actually lies outside of the SAS project file. For example, the Create_data program actually resides in the Perm folder as C:\perm\create_data.sas. Linked programs can be created simply by dragging a SAS program file into the SAS Enterprise Guide workspace or by selecting "Open Program" and choosing a specific program to import into the project.

Reusability is greatly facilitated through linked programs because external SAS programs can be modified in only one location and affect all subsequent uses of the code. For example, The Macros program is currently an imbedded program that defines macros used in this project. However, if external SAS programs or projects could benefit from the same macros, the code must be inefficiently copied or imported from the current project, thwarting reusability. Thus, a preferred design to promote software reuse would incorporate an external SAS program (e.g., C:\perm\macros.sas) that could be linked to this and other programs or projects. Moreover, software maintainability is improved because the external program can be updated in a single place to affect all other uses of its code.

A potential downside of centralized software reuse can be decreased security. Were Macros.sas maintained in a single, external SAS program and linked to (rather than imbedded in) this project (or otherwise referenced via the %INCLUDE statement or SAS Autocall Macro Facility), control over that code might be wrested from the programmer's grasp. Rather than being owned by a single individual and utilized in a stovepipe, the program would be owned by the team or organization and possibly controlled via change control policies and procedures. Thus, where code reuse is supported in production software, measures must be taken to ensure that linked or referenced modules are not accidentally modified or modified intentionally with unintended consequences. Reuse libraries and catalogs facilitate this objective by documenting and organizing software assets, including those intended for reuse.

While Figure 1 depicts the view of project files that most users see, by renaming the file from an EGP to a ZIP extension, the structure of the project file can be understood and interrogated. Figure 2 demonstrates the same project file after it has been renamed and opened with a zip application such as WinZip or WinRAR.

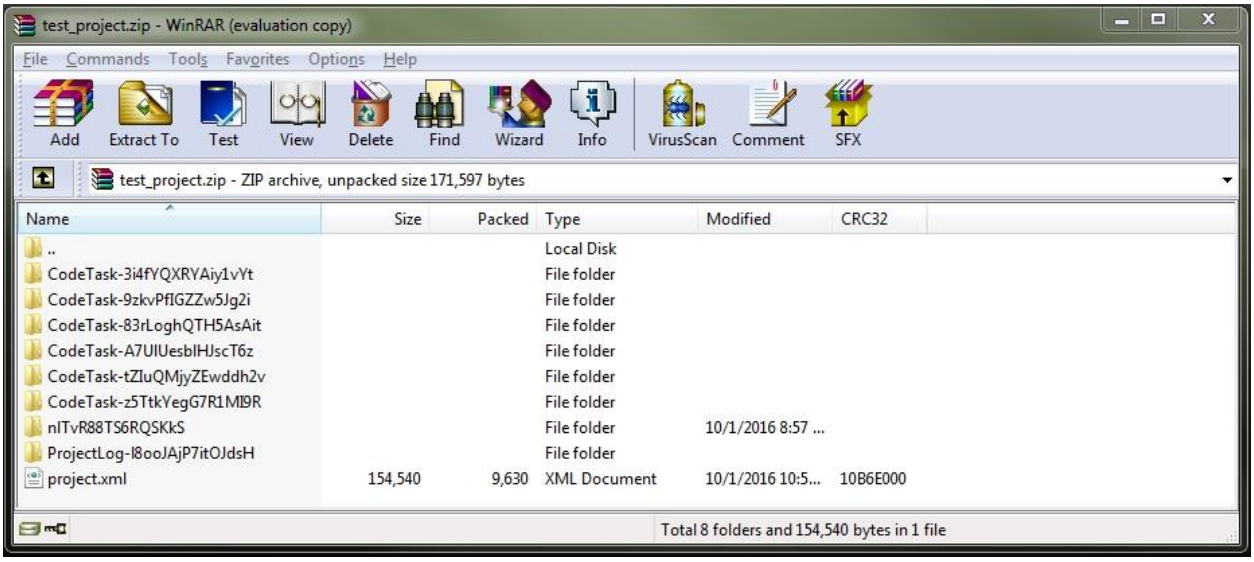

Figure 2. View of a SAS Enterprise Guide Project File as a Zipped Folder Structure

Within each project file, Project.xml is the hub of information. Metadata about project content, linked or imbedded programs, SAS data sets, process flow relationships, notes, last project execution date, and other information is all stored in this file. The READXML macro exploits the structure of the XML file to identify all imbedded and linked programs and to ingest all SAS notes. For example, all SAS programs are identified by the <Element Type="SAS.EG.ProjectElements.CodeTask"> tag while the corresponding <Label> tag represents the program name and the <ID> tag represents the program ID within the XML structure. Thus, the following XML snipped demonstrates that that imbedded SAS program Initialize can be found in the zipped folder CodeTask-9zkvPfIGZZw5Jg2i:

```
<Element Type="SAS.EG.ProjectElements.CodeTask">
    <Element>
       <Label>initialize</Label>
       <Type>TASK</Type>
       <Container>PFD-Ti6KbIpksie4aNSb</Container>
       <ID>CodeTask-9zkvPfIGZZw5Jg2i</ID>
```

Figure 3 demonstrates the XML snippet in which the imbedded program Initialize is referenced. Note that other metadata (not described or utilized in this text) such as create date and modify date are also maintained within the SAS Enterprise Guide metadata structure.

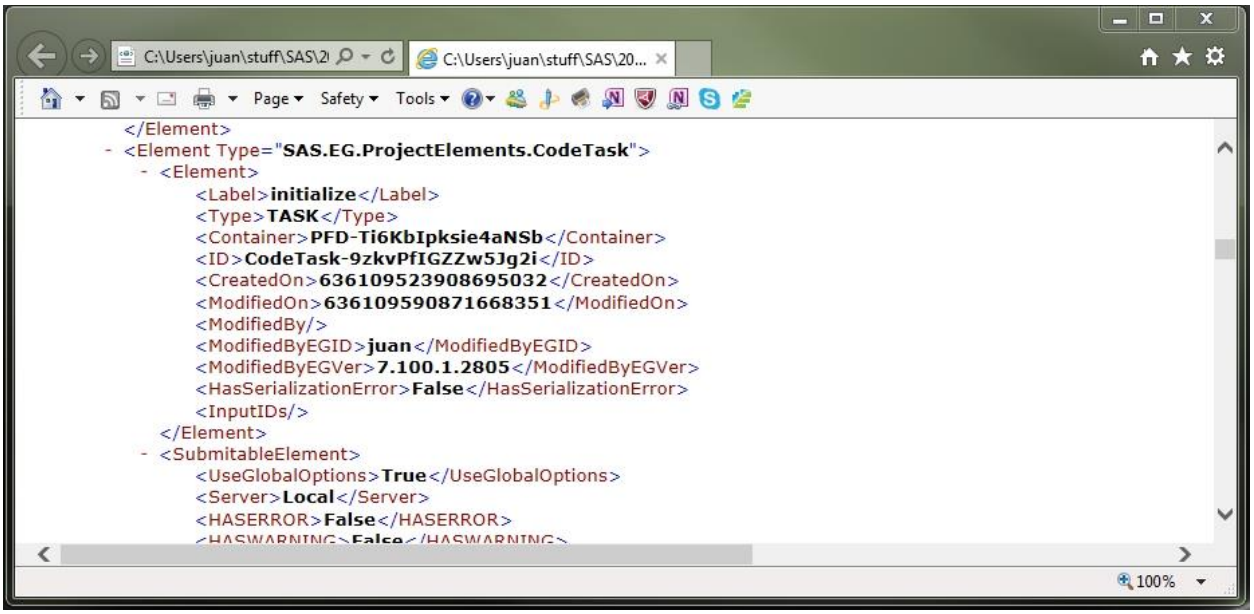

Figure 3. View of a SAS Enterprise Guide Project.xml File (Imbedded SAS Program)

With the zipped folder now identified in which Initialize resides (i.e., CodeTask-9zkvPfIGZZw5Jg2i), the user (or, in this case, the READCODE macro) can navigate to that folder and extract the contents of the Initialize program. Figure 4 demonstrates the zipped view of CodeTask-9zkvPfIGZZw5Jg2i folder with Initialize renamed as code.sas. Within the SAS Enterprise Guide compressed folder structure, all programs are saved as code.sas, thus the program ID is required to identify actual program names and where specific programs are located.

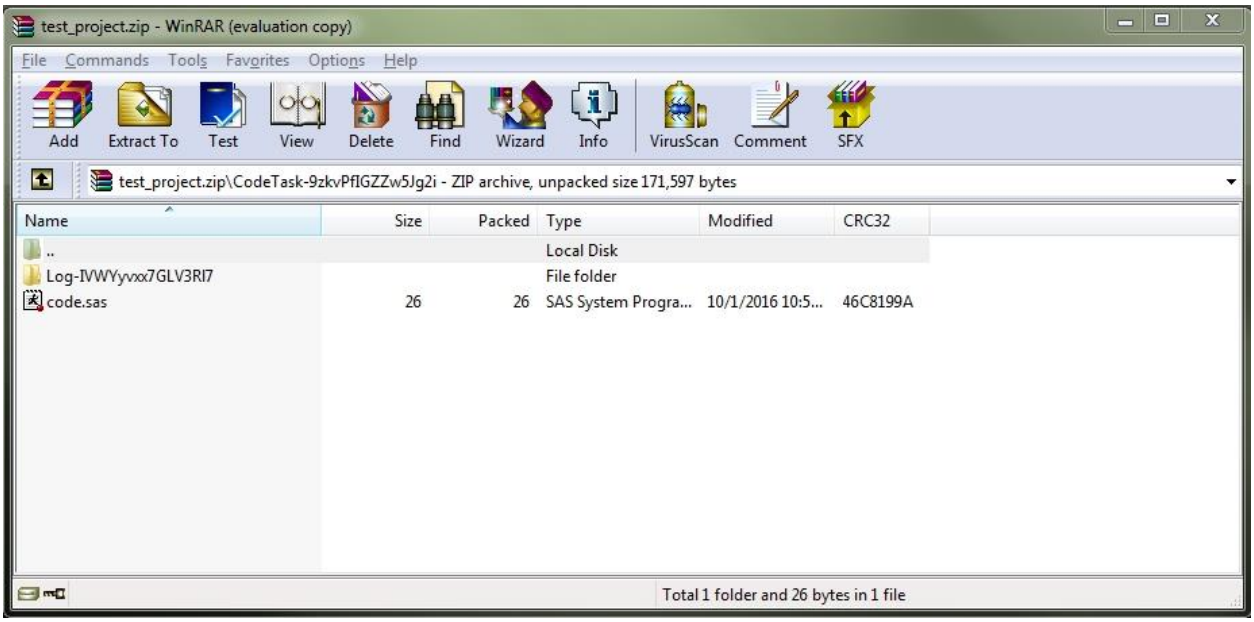

Figure 4. View of Initialize Program Within Compressed Folder Structure

To identify linked programs, the <DNA> tag (subsumed under <CodeTask>) can be utilized to determine the location of the linked program. For example, further interrogating the Project.xml file demonstrates that the Create_data program has an ID of CodeTask-A7UlUesblHJscT6z as shown in Figure 5.

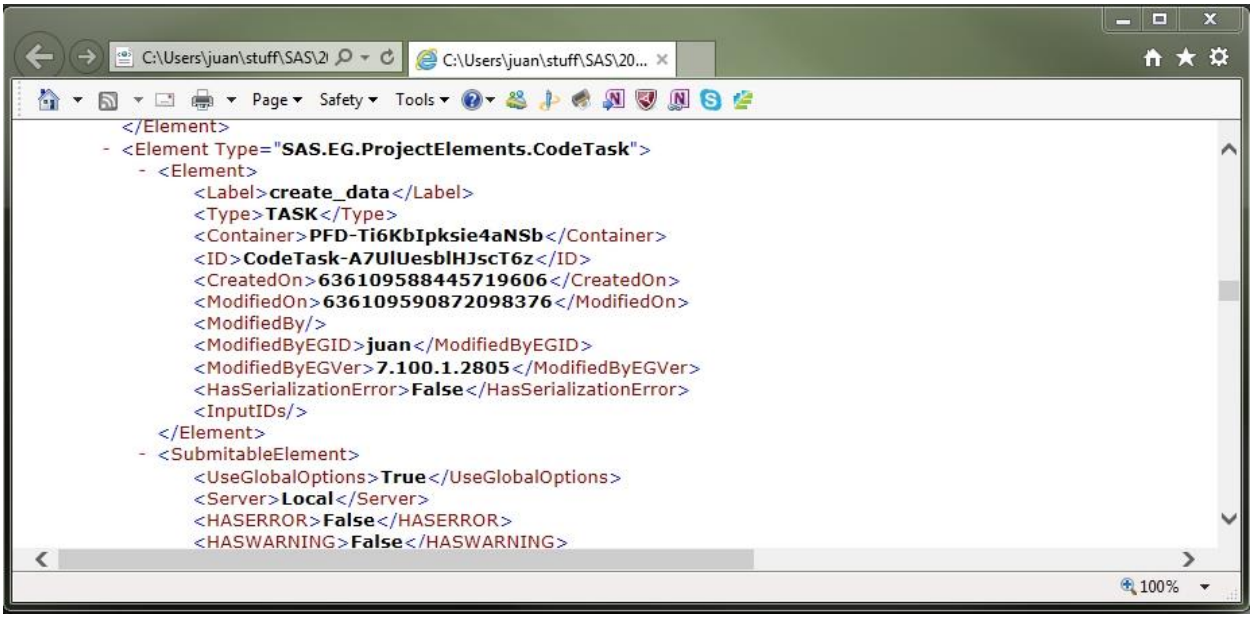

Figure 5. View of a SAS Enterprise Guide Project.xml File (Linked SAS Program)

However, because the SAS program is linked—not imbedded—no Code.sas program file resides within the corresponding zipped folder. The external code instead must first be located within the <DNA> tag, which is demonstrated in Figure 6 and shows the program is in fact C:\perm\create_data.sas.

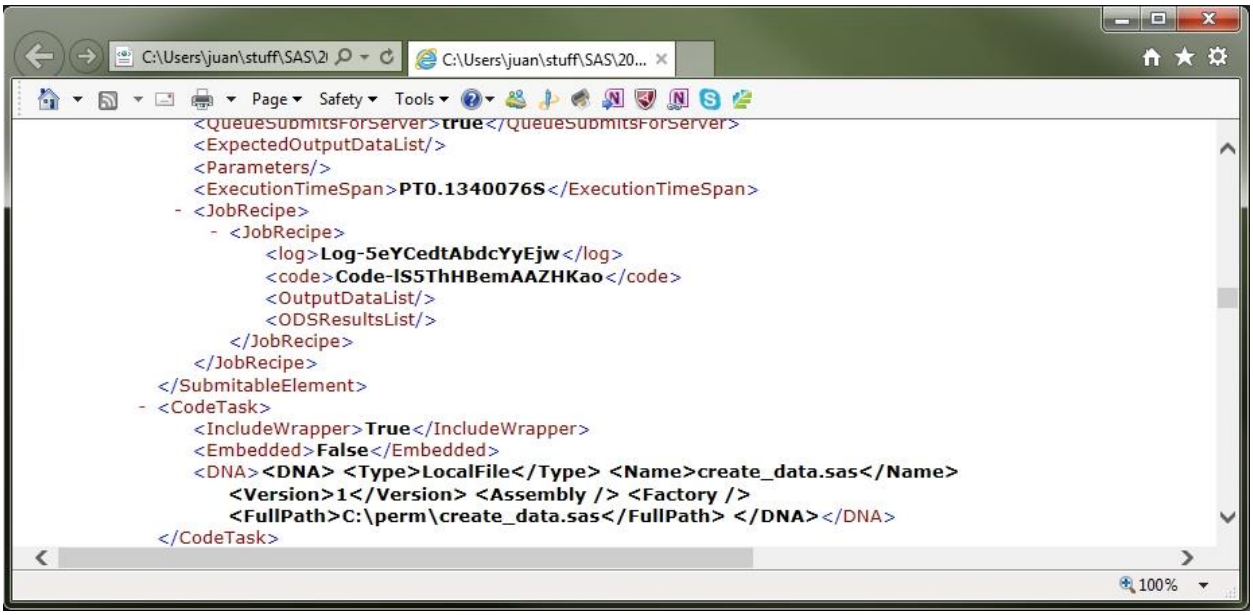

Figure 6. View of a SAS Enterprise Guide Project.xml File (Linked SAS Program)

With the program now located, READXML terminates and the READCODE macro ingests and parses the Create_data program. Both linked and imbedded SAS programs within SAS Enterprise Guide project files, as well as externally saved SAS programs, are parsed with the same READCODE macro. After all, a text touting reuse and reusability principles should actually demonstrate macro flexibility and reuse.

SETUP AND STRUCTURE

The SCAVENGER macro requires little setup—only a configuration file (demonstrated later). In this example, the configuration file (C:\perm\scavenger.cfg) is used, so the Perm folder is referenced. The macro definition follows:

```
%macro scavenger(dir= /* asterisk delimited list of folders to include */,
    cfg= /* folder and name of configuration file */,
    dsnout= /* metadata data set in LIB.DSN format (which is first deleted) */,
    subdir=YES /* YES to parse subdirectories, NO to not */,
    filetype=SAS EGP /* space-delimited list of all file extensions to find */);
```

If the SUBDIR parameter is omitted, all subdirectories subsumed under the folder specified in the DIR parameter will also be included in the search. If the FILETYPE parameter is omitted, all SAS project and program files will be included in the search. For example, a sample invocation of SCAVENGER follows:

```
%scavenger(dir=c:\perm, cfg=c:\perm\scavenger.cfg, dsnout=metadata);
```

The configuration file is ingested only once with the READCONFIG macro, called within the READCODE macro. The configuration file contains a list of <bracketed> tags, with each tag followed by its description. The tags are dynamically created as SAS variables and thus must conform to SAS variable naming conventions. The descriptions that follow the tags become SAS labels, thus they must conform to SAS labeling conventions. These rules are not enforced through exception handling, so failure to perform manual quality control on configuration files will cause program failure. All lines that do not begin with <bracketed> terms are ignored and thus can be used as comments or instructions for creation of the configuration file. For example, the following sample configuration file prescribes its own rules and can be saved as C:\perm\scavenger.cfg:

```
* IN THIS CONFIGURATION FILE...

* <bracketed> terms represent tags that are parsed into comma-delimited variables
* tags cannot contain spaces and must conform to SAS variable naming conventions
* tag metadata must conform to SAS labeling conventions
* tag metadata cannot contain asterisks, commas, or other special characters

<desc>Program Description
<ver>Program Version
<auth>Author
<crdt>Create Date
<updt>Last Update
<test>Test Date
<vadt>Validation Date
<prdt>Production Date
<reqver>Required SAS Version
<vuln>Vulnerabilities

* WHEN THESE TAGS ARE USED IN SAS PROGRAMS...

* the final character (i.e., semicolon) of all metadata is removed

* an asterisk must precede each tag so author would appear as *<auth>Art C.;
* all metadata are ingested as 100 character text fields and saved as 500 character
variables
* for example, five <desc> tags could be used to create a longer description

* in the data set that is created, TAG_ precedes all variables to avoid collision
* for example, the <auth> tag is saved as the 500 character variable TAG_AUTH
```

The SCAVENGER macro calls other macros, including PARSEDIR, READXML, READCODE, and READCONFIG, all of which are included in Appendix A:

- **PARSEDIR—**This macro iteratively parses a directory (and optionally its subdirectories) to create a data set (DSN parameter) that includes the folder location, file name, and create date for each identified file that matches a space-delimited list of acceptable file extensions. For example, the default FILETYPE parameter is SAS EGP, representing that PARSEDIR will identify and analyze only SAS program and project files. The macro is called only once with the following definition:

```
%macro parsedir(dir= /* logical location of folder to parse */,
    dsn= /* data set in LIB.DSN format into which to put directory info */,
    subdir=YES /* YES to parse subdirectories, NO to only parse the current DIR */,
    filetype=SAS EGP /* space-delimited list of file extensions to find */);
```

- **READXML—**This macro is dynamically invoked whenever SCAVENGER encounters a SAS project file. All SAS Enterprise Guide project files are compressed zip files, each of which contains the file Project.xml in its root structure. Thus, by peering into the zipped project file (with the ZIP option of the FILENAME statement), READXML interrogates the Project.xml file and extracts information, including the names of all imbedded SAS programs, the names of all linked SAS programs, and a SAS project note (only one note) if it exists within the project. With this information, READCODE then iteratively opens and parses each imbedded SAS program (skipping the linked programs). For each SAS Enterprise Guide project file that is encountered, READXML is called only once with the following definition:

```
%macro readxml(egpfile= /* folder and name of EGP file */,
    xmlfile=project.xml /* name of XML inside EGP file */);
```

- **READCODE—**This macro ingests and parses SAS programs, including both externally saved SAS programs and imbedded SAS programs that are codified within SAS Enterprise Guide project files. The FILENAME statement, called dynamically from SCAVENGER, enables this flexibility. When READCODE encounters the first program file it reads the configuration file to retrieve a list of dynamic tags that will be searched for in SAS programs. For example, including <auth>Author in the configuration file will cause READCODE to search for this tag in all SAS programs and extract the associated metadata that follows. In addition to parsing tags dynamically generated from the configuration file, READCODE also assesses other metadata, including all macros that are defined, all user-created SAS global macro variables, and the number of lines of code. The macro definition follows:

```
%macro readcode(cfg= /* folder and name of configuration file */,
    infilename= /* file reference to the SAS program file */,
    dsn= /* metadata data set in LIB.DSN format */);
```

- **READCONFIG—**This macro is called only when READCODE is called for the first time, when encountering the first SAS program file. A sample configuration file is demonstrated previously, and the macro definition follows:

```
%macro readconfig(cfg= /* folder and name of configuration file */);
```

PROGRAM FLOW AND METADATA

The SCAVENGER program flow begins by calling the PARSEDIR macro to create a data set containing all SAS projects and programs. Thereafter, SCAVENGER enters an outer loop as it iterates through the data set created by PARSEDIR. Because subsequent DATA steps are required inside this loop, input/output (I/O) functions such as OPEN, FETCHOBS, GETVARC, and GETVARV are utilized to iterate through the data set without use of the DATA step. When a SAS program file is encountered, the READCODE macro is invoked.

When a SAS project file is encountered, however, SCAVENGER interrogates the zipped project file with the READXML macro to determine the number (and name) of all imbedded and linked SAS programs held inside the project file. When the XML file has been read, SCAVENGER enters a second, internal loop and iterates through each

program file. Imbedded program files are ingested in this second loop with the READCODE macro, while linked program files are not ingested but are updated (in the metadata data set) with the project name, program name, project file creation date, and single project note (if it exists). To help further distinguish imbedded programs from linked programs, all linked program file names are not included in the file name column but rather in the INCLUDELIST column, since the link references the SAS program similar to use of the %INCLUDE statement.

When a SAS program file (rather than a project file) is encountered in the first, outer loop, SCAVENGER also calls the READCODE macro to parse and analyze the SAS program. Regardless of whether READCODE is called for imbedded SAS program files or externally saved SAS program files, READCODE creates a one-observation data set for each program that it encounters and continually appends this observation to results from previous iterations. Thus, READCODE incrementally builds the metadata data set that drives reuse catalog generation, as well as other reporting or analysis that might be conceptualized. In this example, the Metadata data set is created in the WORK directory, which includes the following variables:

- **SOFTWARENAME** — This variable is the full path and name (with extension) of the SAS project or program file. Because the path is included, this variable can serve as the unique key when joining this table for other purposes.

- **PROGRAMNAME** — This variable is the abbreviated name of the program, thus without the path or extension. For SAS program files (that are saved externally in a folder), the PROGRAMNAME will always be identical to the SOFTWARENAME, with only the path and extension removed. However, for imbedded SAS programs (inside SAS Enterprise Guide project files), PROGRAMNAME will be the name of imbedded SAS program. For linked SAS programs (also inside project files), PROGRAMNAME will be blank, representing that the program was not interrogated, since it exists externally and may have already been analyzed separately.

- **EGPNOTE** — The SAS Enterprise Guide Note contains the first 500 characters of a project note. Because these notes are endemic to SAS Enterprise Guide, this variable is blank for all externally observed SAS programs. Moreover, because each note references the project file itself (and not an individual SAS program therein), EGPNOTE is ascribed to all program files (imbedded and linked) found within a specific project file.

- **SAVEDATE** — The date (in DATETIME17 format) that the file was saved is generated through the FILENAME statement within the PARSEDIR macro. Externally saved SAS program files will have individual date-time stamps, while SAS programs within SAS Enterprise Guide project files will inherit the date-time stamp from the project file itself.

- **MACROLIST** — This space-delimited field represents a list of all macro definitions that appear in a program. The field is useful because with little effort it can be used to demonstrate redundancy, dependency, conflicts, or other interaction. For example, if an identically named macro is defined in multiple places, this could represent redundant code that should be ingested into a shared repository like a reuse library. However, if the redundantly named macros perform disparate functions, this could cause conflict (and failure) were the wrong macro ever incorrectly referenced due to this confusion.

- **INCLUDELIST** — This asterisk-delimited field represents a list of all program dependencies in which the %INCLUDE statement is utilized to reference external code. This field can be used to detect dependencies between programs as well as to ensure that reuse of software (via %INCLUDE) is pointing at the most complete, correct, or recent version of some SAS program or saved macro. Moreover, when a macro or other reusable code module must be modified, examination of the INCLUDELIST field will immediately identify all software products that rely on a specific external program or macro. With this information, backward compatibility and regression testing are tremendously facilitated.

- **GLOBALLIST** — This space-delimited field represents all user-defined global macro variables that are explicitly defined with the %GLOBAL statement, including both those that are defined singly and in series. Because global macro variables persist between SAS macros and even SAS programs running in the same SAS session, they represent a tremendous security risk when not implemented with consistency and care. For example, if two macros each define the global macro variable &MYMACROVAR, each macro will

overwrite the macro variable value of the other. To help identify macro variable conflicts that could unsuspectingly topple software, a comparison of all global macro variables within a SAS infrastructure or environment (not demonstrated) could substantially mitigate this risk.

- **CODELINES** — This variable represents the number of lines of code in each externally saved or internally imbedded SAS program. Whereas this is an easy feat (aided by the <CTRL> and <END> keys) in any SAS program, the girth of SAS project files is sometimes more difficult to calculate because it can involve opening dozens of separate program files. Thus, by summing the lines of code across all imbedded SAS programs within a SAS project, an accurate picture of the size of the project can be immediately ascertained.

- **!!! DYNAMIC VARIABLES !!!** — While the previous variables are produced in all metadata data sets, additional variables can be dynamically added, enabling search and documentation of code to be highly customized. For example, given the tags and descriptions provided in the previous sample configuration file (Scavenger.cfg), the following dynamic variables would be appended to the metadata data set, with each variable representing a concatenation of all instances in which a specific tag was observed: TAG_DESC, TAG_AUTH, TAG_CRDT, TAG_UPDT, TAG_TEST, TAG_VADT, TAG_PRDT, TAG_REGVER, and TAG_VULN. As a default, each dynamic variable is truncated at 500 characters. For example, if one tag (<desc>this is a two-line;) occurs on line 10 and a second tag (<desc>program description.;) is found on line 52, the TAG_DESC variable will be set to “this is a two-line program description.”

SAMPLE OUTPUT

The following output demonstrates execution of SCAVENGER on three files: the sample project file depicted in Figure 1, the Create_data program linked from the project file, and the Scavenger program itself. When these three files are located in the C:\perm folder, Figure 7 demonstrates the output of the PARSEDIR macro.

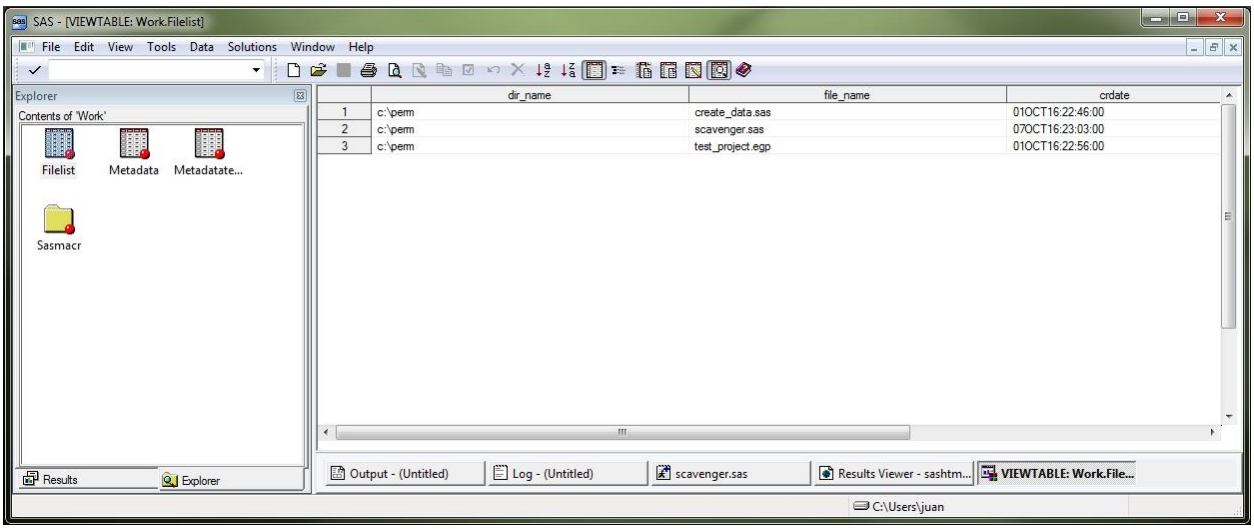

Figure 7. Output Data Set (Filelist) of PARSEDIR Macro

SCAVENGER continues to execute after the PARSEDIR macro and the READCODE macro ingests and parses the various imbedded, linked, and external SAS program files. READCODE ultimately creates the Metadata data set, partially depicted in Figure 8, which is the substrate from which reuse catalogs and other automated artifacts can be constructed.

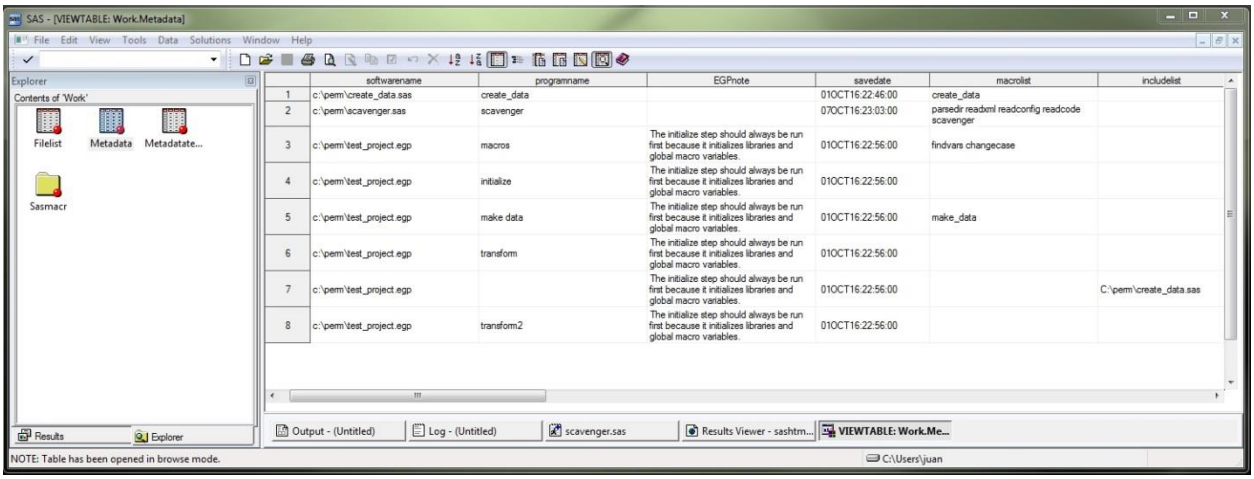

Figure 8. Output Data Set (Metadata) of READCODE Macro

For example, executing the following REPORT procedure generates an extremely rudimentary reuse catalog that can be used to identify project and program names, locations, dependencies, and global macro variables that are created:

```
proc report data=metadata;
    column softwarename programname savedate macrolist globallist;
run;
```

The output of the REPORT procedure is demonstrated in Figure 9.

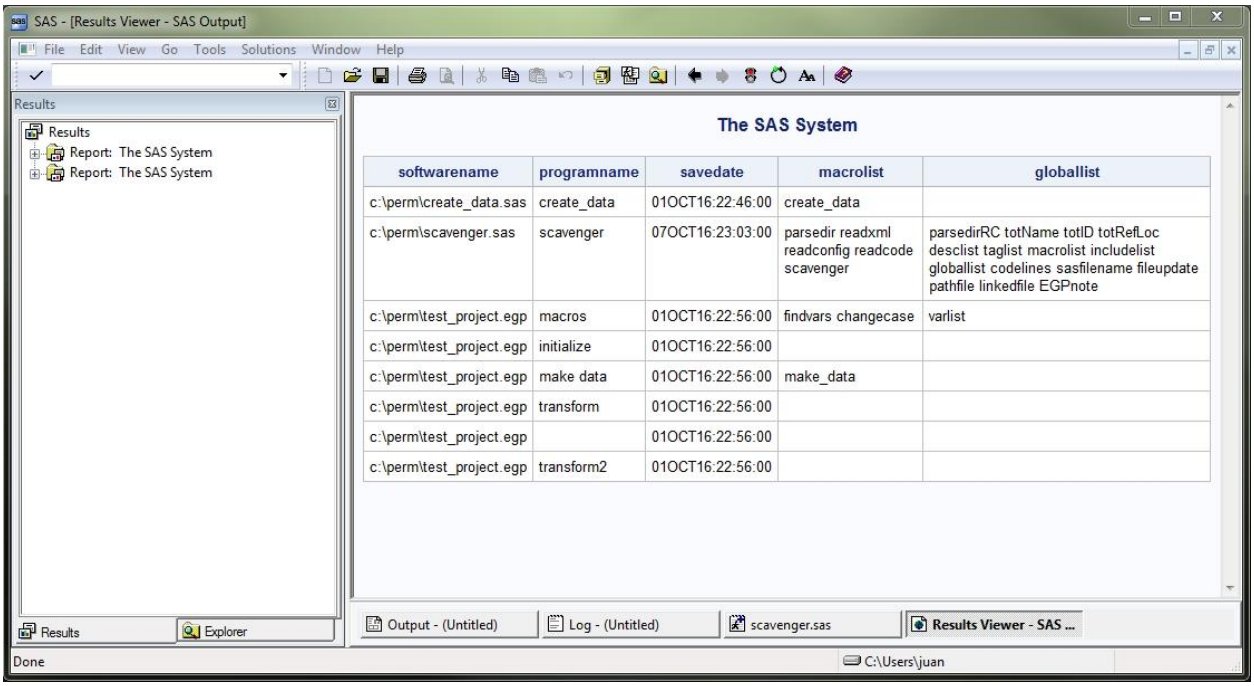

Figure 9. Output from REPORT Procedure Run on Metadata Data Set

The simplicity of the output facilitates readability but this example represents just one of many potential reports that could be generated from the underlying Metadata data set. The PROGRAMNAME variable is blank for the 7th observation, representing the linked Create_data program found within the SAS Enterprise Guide project. If the report contained the INCLUDELIST variable, it would indicate “c:\perm\create_data.sas” for the 7th observation.

FUTURE STEPS

The SCAVENGER macro suite is a scalable solution that produces a standardized data set that represents an organization's software infrastructure, including all SAS project and program files. While not demonstrated, the metadata results can be summarized at the project, program, or macro level to produce reuse catalogs through which SAS users can better organize, understand, and search code. Future SCAVENGER enhancements could include:

- **Deconflict Global Macro Variables** — With a simple DATA step and subsequent REPORT procedure, all global macro variables within an infrastructure could be compared, demonstrating where identically named macro variables are in conflict. For an even more robust report, %LET statements could be additionally parsed and analyzed, since global macro variables can also be assigned when the %LET statement is invoked outside of a macro definition.

- **Include Cryptographic Checksums** — Although Base SAS cannot generate checksums on SAS files, the PARSEDIR macro could be revised to shell to the operating system (OS) to run Python or other scripts to produce checksums for each program file. The author demonstrates this functionality in a separate text, enabling SAS practitioners to validate software integrity, thus supporting both security and reusability.[iv]

- **Support Quality and Risk Management** — Information about threats and vulnerabilities can be included in software, saved as standardized SAS comments. For example, in a separate text, the author demonstrates that comments that depict refactoring opportunities can be preceded with three asterisks (***) while comments that depict vulnerabilities that could cause functional or performance failure can be preceded with five asterisks (*****).[v] This standardized commenting enables SAS practitioners to assess the expected quality and caveats of software more accurately (via automated reports) before reusing software modules.

CONCLUSION

The SCAVENGER program expands the ability to interrogate SAS program files to include SAS Enterprise Guide project files and the program files imbedded within them. With this capability, SCAVENGER iteratively and efficiently parses the collective body of shared SAS programs within an environment to produce a SAS data set that includes metadata for all SAS programs. This text demonstrated one remarkable use of SCAVENGER—to create software reuse catalogs that can be used to facilitate and measure software reuse within an organization—but a hundred other uses could be conceptualized to support efficient software development and management within a SAS enterprise.

REFERENCES

[i] ISO/IEC 25010:2011. *Systems and software engineering—Systems and software Quality Requirements and Evaluation (SQuaRE)—System and software quality models*. Geneva, Switzerland: International Organization for Standardization and Institute of Electrical and Electronics Engineers.

[ii] IEEE Std 1517-2010. *IEEE standard for information technology—System and software life cycle processes—Reuse processes*. Geneva, Switzerland: Institute of Electrical and Electronics Engineers.

[iii] Id.

[iv] Hughes, Troy Martin. *SAS Data Analytic Development: Dimensions of Software Quality*. New York, NY: John Wiley and Sons Publishing, 2016.

[v] Id.

CONTACT INFORMATION

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

Name: Troy Martin Hughes
E-mail: troymartinhughes@gmail.com

APPENDIX A. SCAVENGER SOFTWARE

```
%macro parsedir(dir= /* logical location of folder to parse */,
    dsn= /* data set in LIB.DSN format into which to put directory info */,
    subdir=YES /* YES to parse subdirectories, NO to only parse the current DIR */,
    filetype=SAS EGP /* space-delimited list of file extensions to find */);
%let syscc=0;
%global parsedirRC;
%let parsedirRC=GLOBAL FAILURE;
%local subdirs i ext;
%if %upcase(&subdir)=YES %then %let subdirs=/s;
%else %let subdirs=;
filename filelist pipe "dir &subdirs &dir";
data &dsn (keep=dir_name file_name crdate);
    length dir_name $200 file_name $200 crdate 8 inp $400;
    format crdate datetime17.;
    infile filelist truncover;
    input inp $400.;
    if length(inp)>9 and substr(strip(inp),1,9)='Directory' then
dir_name=substr(strip(inp),13);
    else if find(inp,'.')>0 then do;
       if
    %let i=1;
    %do %while(%length(%scan(&filetype,&i,,S))>1);
       %let ext=%scan(&filetype,&i,,S);
          %if &i>1 %then or;
             lowcase(scan(inp,-1,.))="%lowcase(&ext)"
          %let i=%eval(&i+1);
          %end;
       then do;
          crdate=input(substr(inp,1,20),mdyampm20.);
          file_name=lowcase(substr(inp,39));
          output;
          end;
       end;
    retain dir_name;
run;
%if &syscc=0 %then %let parsedirRC=;
%mend;

%macro readxml(egpfile= /* folder and name of EGP file */,
    xmlfile=project.xml /* name of XML inside EGP file */);
%global totName totID totRefLoc;
%let totName=;
%let totID=;
%let totRefLoc=;
%let EGPnote=;
filename projzip zip "&egpfile" member="&xmlfile";
data _null_;
    infile projzip truncover encoding='utf-16be';
    input line $400.;
    length foundCode 8 foundElement 8 foundCodeTask 8 foundDNA 8
       programName $100 programID $40 programRefLoc $200
       totName $1000 totID $1000 totRefLoc $1000 sep $2
       EGPnote $1000;
```

```
    retain foundCode foundElement foundCodeTask foundDNA foundNote re
       programName programID programRefLoc totName totID totRefLoc EGPnote;
    if _n_=1 then do;
       foundCode=0;
       foundElement=0;
       foundCodeTask=0;
       foundDNA=0;
       foundNote=0;
       programName='';
       programID='';
       programRefLoc='';
       totName='';
       totID='';
       totRefLoc='';
       re=prxparse("s/<.*?>//");
       end;
    line=strip(line);
    if line=:'<Element Type="SAS.EG.ProjectElements.CodeTask">' then foundCode=1;
*starts element;
    else if foundCode and line=:'<Element>' then foundElement=1;
    else if foundCode and foundElement then do;
        if line=:'<Label>' then call prxchange(re,-1,line,programName);
        else if line=:'<ID>' then call prxchange(re,-1,line,programID);
        else if line=:'</Element>' then foundElement=0;
       end;
    else if foundCode and not foundElement then do;
        if not foundCodeTask then do;
            if line=:'<CodeTask>' then foundCodeTask=1;
            else if line=:'</Element>' then do; * terminates element;
              sep=ifc(length(totName)=1,'','*');
              totName=strip(totName) || sep || strip(programName);
              totID=strip(totID) || sep || strip(programID);
              totRefLoc=strip(totRefLoc) || sep || strip(programRefLoc);
              call symput('totName',strip(totName));
              call symput('totID',strip(totID));
              call symput('totRefLoc',strip(totRefLoc));
              foundCode=0;
              output;
              programName='';
              programID='';
              programRefLoc='';
              end;
          end;
        else if foundCodeTask then do;
            if line=:'<DNA>' then foundDNA=1;
            else if foundDNA and line=:'&lt;FullPath&gt;' then do;
                programRefLoc=substr(line,17,length(line)-34);
              end;
            else if line=:'</CodeTask>' then do;
                foundCodeTask=0;
                foundDNA=0;
              end;
          end;
       end;
```

```
    if line=:'<Element Type="SAS.EG.ProjectElements.Note">' then foundNote=1;
*found a note;
    else if foundNote and line=:'<Text>' then do;
        call prxchange(re,-1,line,EGPnote);
        call symput('EGPnote',strip(EGPnote));
        foundNote=0;
        end;
run;
%mend;

%macro readconfig(cfg= /* folder and name of configuration file */);
%global desclist;
    filename cfgfile "&cfg";
    %if &sysfilrc=0 %then %do;
        data _null_;
            infile cfgfile truncover end=eof;
            length tags $500 desc $1000;
            input line $100.;
            if _n_=1 then do;
                tags='';
                desc='';
                end;
            if find(line,'<')>0 and find(line,'>')>0 and line=:'<' then do;
                tags=catx(' ',tags,'TAG_'||substr(line,2,find(line,'>')-2));
                desc=catx('*',desc,substr(line,find(line,'>')+1));
                end;
            if eof then do;
                call symput('taglist',strip(tags));
                call symput('desclist',strip(desc));
                end;
            retain tags desc;
        run;
        %end;
%mend;


%macro readcode(cfg= /* folder and name of configuration file */,
    infilename= /* file reference to the SAS program file */,
    dsn= /* metadata data set in LIB.DSN format */);
%global taglist macrolist includelist globallist codelines;
%local i var;
* configuration file is read only the first time READCODE is called;
%if %length(&taglist)=0 and %length(cfg)>0 %then %readconfig(cfg=&cfg);
data &dsn.temp (keep=softwarename programname EGPnote savedate macrolist
        includelist globallist codelines
        %if %symexist(taglist) and %symexist(desclist) %then %do;
            %let i=1;
            %do %while(%length(%scan(&taglist,&i,,S))>0);
                %scan(&taglist,&i,,S)
                %let i=%eval(&i+1);
                %end;
            %end;);
    infile &infilename truncover end=eof;
    length line $1000 softwarename $200 programname $200 EGPnote $1000
```

```
            savedate 8 macrolist $500 includelist $2000 globallist $500 codelines 8
         %if %symexist(taglist) and %symexist(desclist) %then %do;
            %let i=1;
            %do %while(%length(%scan(&taglist,&i,,S))>0);
               %scan(&taglist,&i,,S) $500
               %let i=%eval(&i+1);
               %end;
            %end;;
      format savedate datetime17.;
%if %length(&sasfilename)>0 %then %do; * to ingest the SAS program;
      input line $1000.;
      if _n_=1 then do;
         macrolist='';
         includelist='';
         globallist='';
         codelines=0;
         %if %symexist(taglist) and %symexist(desclist) %then %do;
            %let i=1;
            %do %while(%length(%scan(&taglist,&i,,S))>0);
               %scan(&taglist,&i,,S)='';
               %let i=%eval(&i+1);
               %end;
            %end;
         * because 3 characters precede the first line in EG imbedded programs;
         if substr(line,1,1)=byte(239) and substr(line,2,1)=byte(187) and
            substr(line,3,1)=byte(191) then line=substr(line,4);
         end;
      codelines=codelines+1;
      * MACRO DEFINITIONS;
      if lowcase(strip(line))=:'%macro' then macrolist=
         catx(' ',macrolist,scan(line,2));
      * MACRO INCLUSION;
      if lowcase(strip(line))=:'%include' then do;
         includelist=catx(' * ',includelist,
                dequote(lowcase(strip(substr(line,find(line,'%include')+8,
                find(line,';')-(find(line,'%include')+8))))));
         end;
      * GLOBAL MACRO VARIABLES;
      if lowcase(strip(line))=:'%global' and find(line,';')>0 then do;
         globallist=catx(' ',globallist,substr(line,find(line,'%global')+8,
            find(line,';')-(find(line,'%global')+8)));
         end;
      * CONFIGURATION FILE TAGS;
      %if %symexist(taglist) and %symexist(desclist) %then %do;
         %let i=1;
         %do %while(%length(%scan(&taglist,&i,,S))>0);
            %let var=%substr(%scan(&taglist,&i,,S),5);
            if lowcase(strip(line))=:"*<&var>" or lowcase(strip(line))=:"* <&var>"
                  then do;
               TAG_&var=catx(' ',TAG_&var,substr(line,find(line,'>')+1,length(line)-
                  1-find(line,'>')));
               end;
            %let i=%eval(&i+1);
            %end;
```

```
        %end;
    retain macrolist includelist globallist codelines
    %if %symexist(taglist) and %symexist(desclist) %then %do;
        %let i=1;
        %do %while(%length(%scan(&taglist,&i,,S))>0);
            %scan(&taglist,&i,,S)
            %let i=%eval(&i+1);
            %end;
        %end;;
    %if %symexist(taglist) and %symexist(desclist) %then %do;
        label
        %let i=1;
        %do %while(%length(%scan(&taglist,&i,,S))>0);
            %scan(&taglist,&i,,S)="%scan(&desclist,&i,*)"
            %let i=%eval(&i+1);
            %end;
        %end;;
    if eof then do;
    %end; * create the following variables even for linked EGP programs;
        softwarename="&pathfile";
        programname="&sasfilename";
        savedate=&fileupdate;
        EGPnote="&EGPnote";
        %if %length(&sasfilename)=0 %then %do;
            includelist="&linkedfile";
            %end;
        output;
    %if %length(&sasfilename)>0 %then %do;
        end;
        %end;
run;
%if %sysfunc(exist(&dsn))=0 %then %do;
    data &dsn;
        set &dsn.temp;
    run;
    %end;
%else %do;
    proc append base=&dsn data=&dsn.temp;
    run;
    %end;
%mend;


%macro scavenger(dir= /* asterisk delimited list of folders to include */,
    cfg= /* folder and name of configuration file */,
    dsnout= /* metadata data set in LIB.DSN format (which is first deleted) */,
    subdir=YES /* YES to parse subdirectories, NO to not */,
    filetype=SAS EGP /* space-delimited list of all file extensions to find */);
%global sasfilename fileupdate pathfile linkedfile EGPnote;
%let sasfilename=;
%let fileupdate=;
%let linkedfile=;
%local i j dsid fil closed;
%if %sysfunc(exist(&dsnout)) %then %do;
```

```
    proc delete data=&dsnout;
    run;
    %end;
%parsedir(dir=&dir, dsn=filelist, subdir=&subdir, filetype=&filetype);
%let dsid=%sysfunc(open(filelist,i));
%if &dsid>0 %then %do;
    %let i=1;
    %do %while(%sysfunc(fetchobs(&dsid,&i))=0);
       %let path=%sysfunc(strip(%sysfunc(getvarc(&dsid,1))));
       %let fil=%sysfunc(strip(%sysfunc(getvarc(&dsid,2))));
       %let fileupdate=%sysfunc(strip(%sysfunc(getvarn(&dsid,3))));
       %let pathfile=&path\&fil;
       %if %lowcase(%scan(&pathfile,-1,.))=egp %then %do;
          %readxml(egpfile=&pathfile, xmlfile=project.xml);
          %let j=1;
          %do %while(%length(%scan(&totName,&j,*))>1);
             %let sasfilename=%scan(&totName,&j,*);
             %if %length(%scan(&totRefLoc,&j,*,m))=0 %then %do; *if
                       imbedded SAS program;
                filename sascode zip "&pathfile"
                       member="%sysfunc(strip(%scan(&totID,&j,*,m)))/code.sas";
                %readcode(cfg=&cfg, infilename=sascode, dsn=&dsnout);
                %end;
             %else %do; * if linked SAS program, the program is not actually read;
                %let sasfilename=;
                %let linkedfile=%scan(&totRefLoc,&j,*,m);
                %readcode(cfg=&cfg, infilename=sascode, dsn=&dsnout);
                %end;
             %let j=%eval(&j+1);
             %end;
          %end;
       %else %if %lowcase(%scan(&pathfile,-1,.))=sas %then %do;
          %let sasfilename=%substr(&fil,1,%length(&fil)-4);
          %let EGPnote=;
          filename sascode "&pathfile";
          %readcode(cfg=&cfg, infilename=sascode, dsn=&dsnout);
          %end;
       %let i=%eval(&i+1);
       %end;
    %let closed=%sysfunc(close(&dsid));
    %end;
%mend;

%scavenger(dir=c:\perm, cfg=c:\perm\scavenger.cfg, dsnout=metadata, subdir=YES,
    filetype=SAS EGP);
```