Improving Security Through Metadata Objects: Permission-Driven Macros

Andrew Gannon
The Financial Risk Group, Cary, NC

Introduction

Abstract

Users can wreak havoc on systems in many different ways – from excessive memory and cpu usage to updating tables with bad data missing data. There are several ways to prevent individuals from doing this – namely by limiting users' access to servers and data. There is a better way to manage permissions for users that is not server or environment specific. By creating a framework of metadata groups and roles, macro specific permissions can be applied at the individual and group levels. This paper seeks to explain and demonstrate how to apply macro level permissions to users via roles and groups in Metadata.

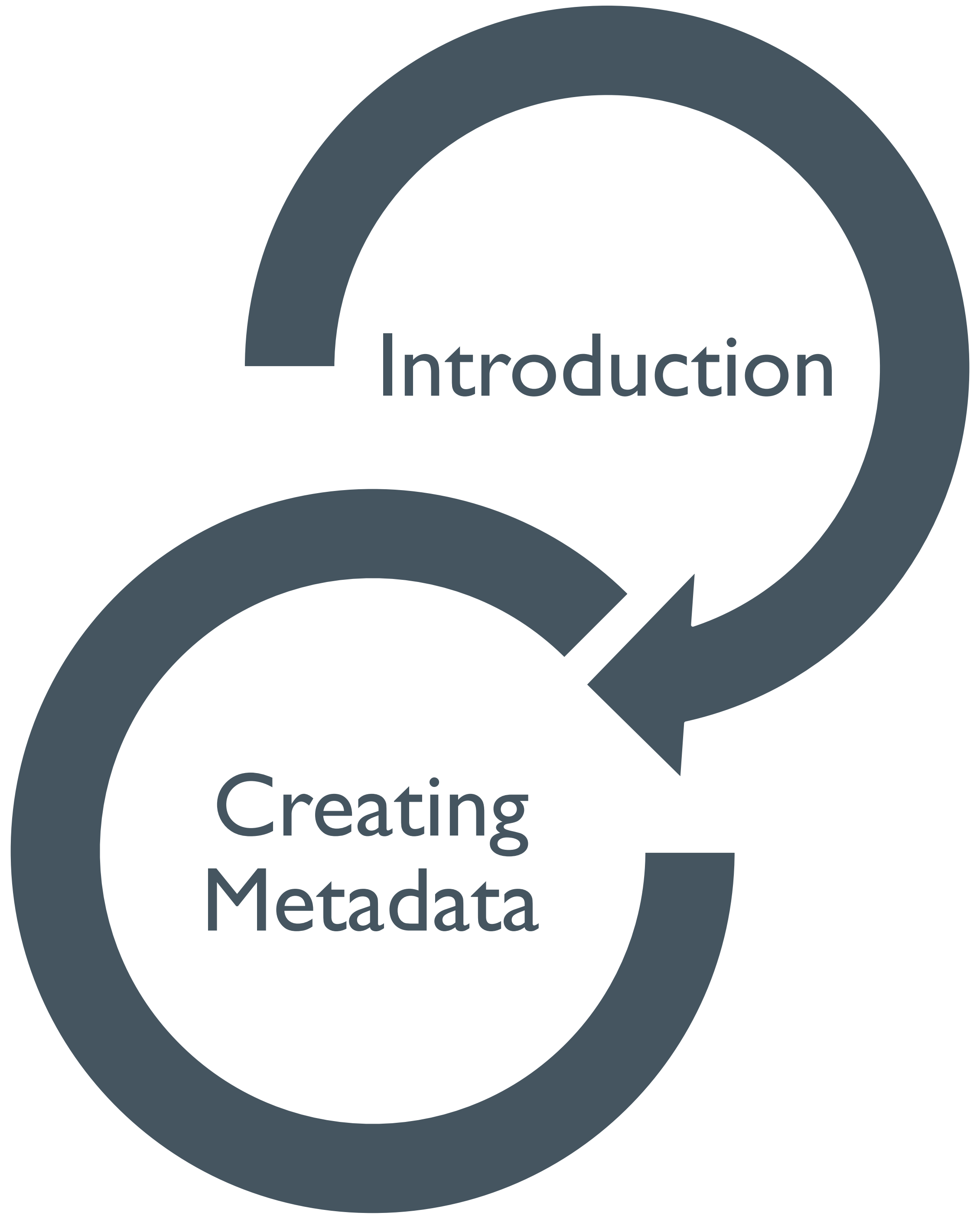
Process Flow

This presentation will describe how system administrators can apply SAS Metadata-based permissions to individual macros utilizing users, roles, and groups in SAS Metadata and apply them with simple SAS code.

Metadata Groups and Roles
Group A
Role 1
Role 2
Role 3
Role 4
SAS Operations
Macro 1
Macro 2
Macro 3
Macro 4

Improving Security Through Metadata Objects: Permission-Driven Macros

Andrew Gannon
The Financial Risk Group, Cary, NC



Introduction

Creating Metadata

SAS Management Console

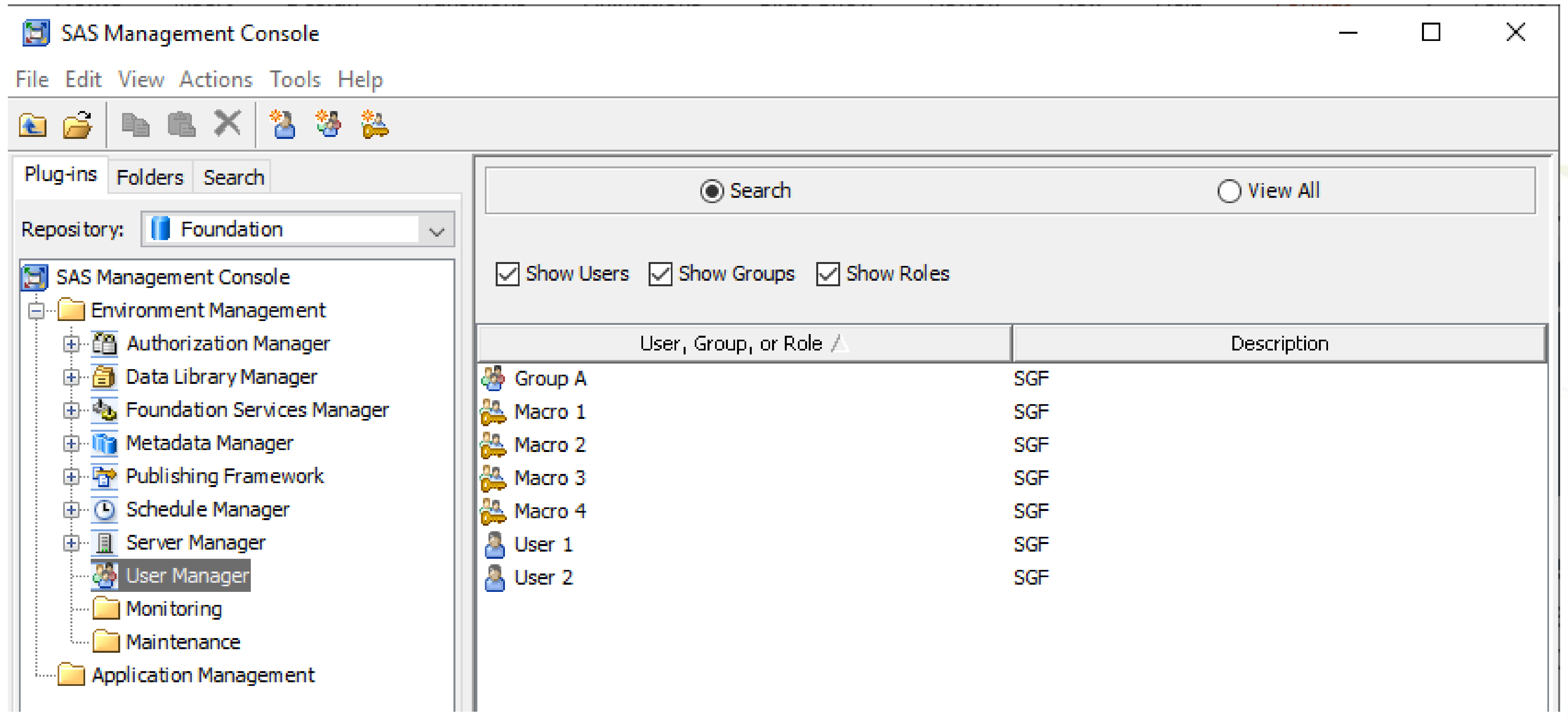
SAS Management Console is the user interface for SAS Metadata. It is the easiest way to create, modify, and delete metadata objects. This is where you can create groups and roles that drive the permissions that this poster discusses.

Roles

SAS Metadata roles are objects that can be assigned to both a user or a group. They are used to control access to application features. When applied to a user, the user has access to the application feature. When applied to a group, the group inherits the role, thereby applying it to any user assigned to the group.

Groups

SAS Metadata groups are logical groupings of roles and groups. They are used to simplify the application of roles. Once a set of roles or groups (sub-groups) are assigned to a group, all the roles and inherited roles from sub-groups are available to any user assigned to the group.

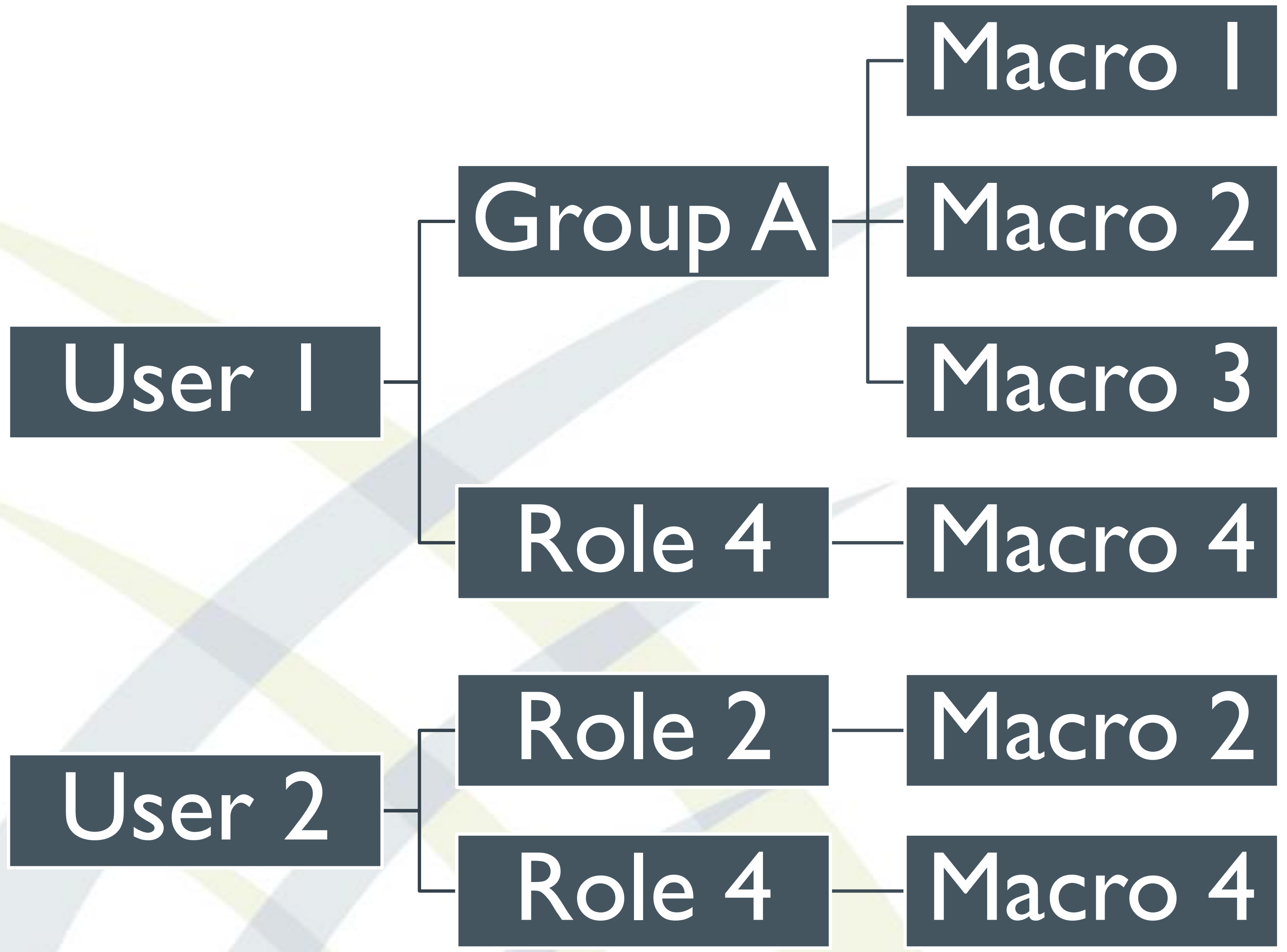


Repository: Foundation

● Search ○ View All

☑ Show Users ☑ Show Groups ☑ Show Roles

User, Group, or Role	Description
Group A	SGF
Macro 1	SGF
Macro 2	SGF
Macro 3	SGF
Macro 4	SGF
User 1	SGF
User 2	SGF



User 1

Group A

Macro 1

Macro 2

Macro 3

Role 4

Macro 4

User 2

Role 2

Macro 2

Role 4

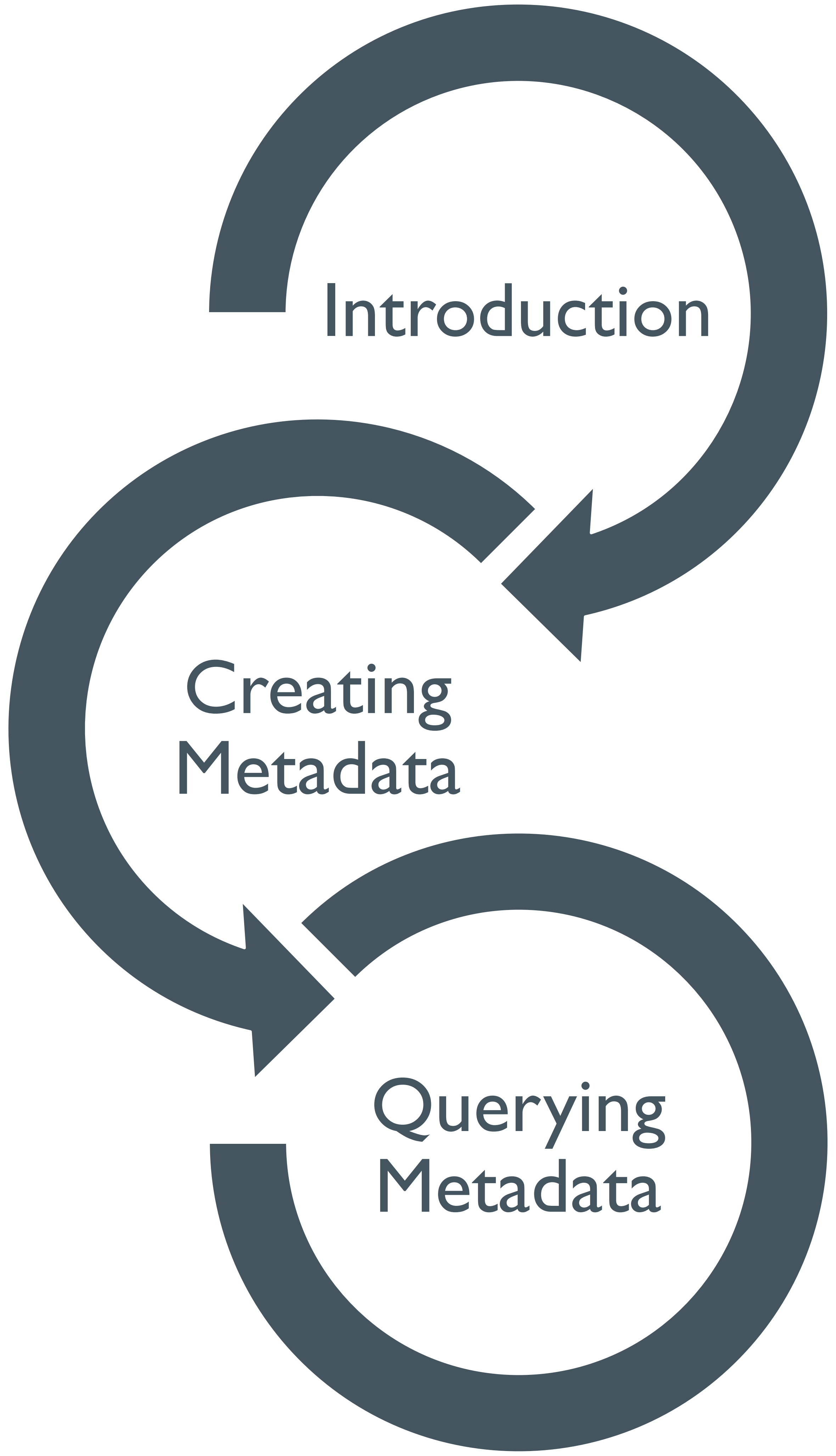
Macro 4

The Framework

The easiest framework for applying permissions is to create a series of roles with same name of the macros. That way the role name can easily be compared to the macro name. Once that is complete, groups can be created to handle similar or logical groupings of roles. Once complete, apply necessary roles and groups to users to finish the metadata portion of permissions application.

Improving Security Through Metadata Objects: Permission-Driven Macros

Andrew Gannon
The Financial Risk Group, Cary, NC

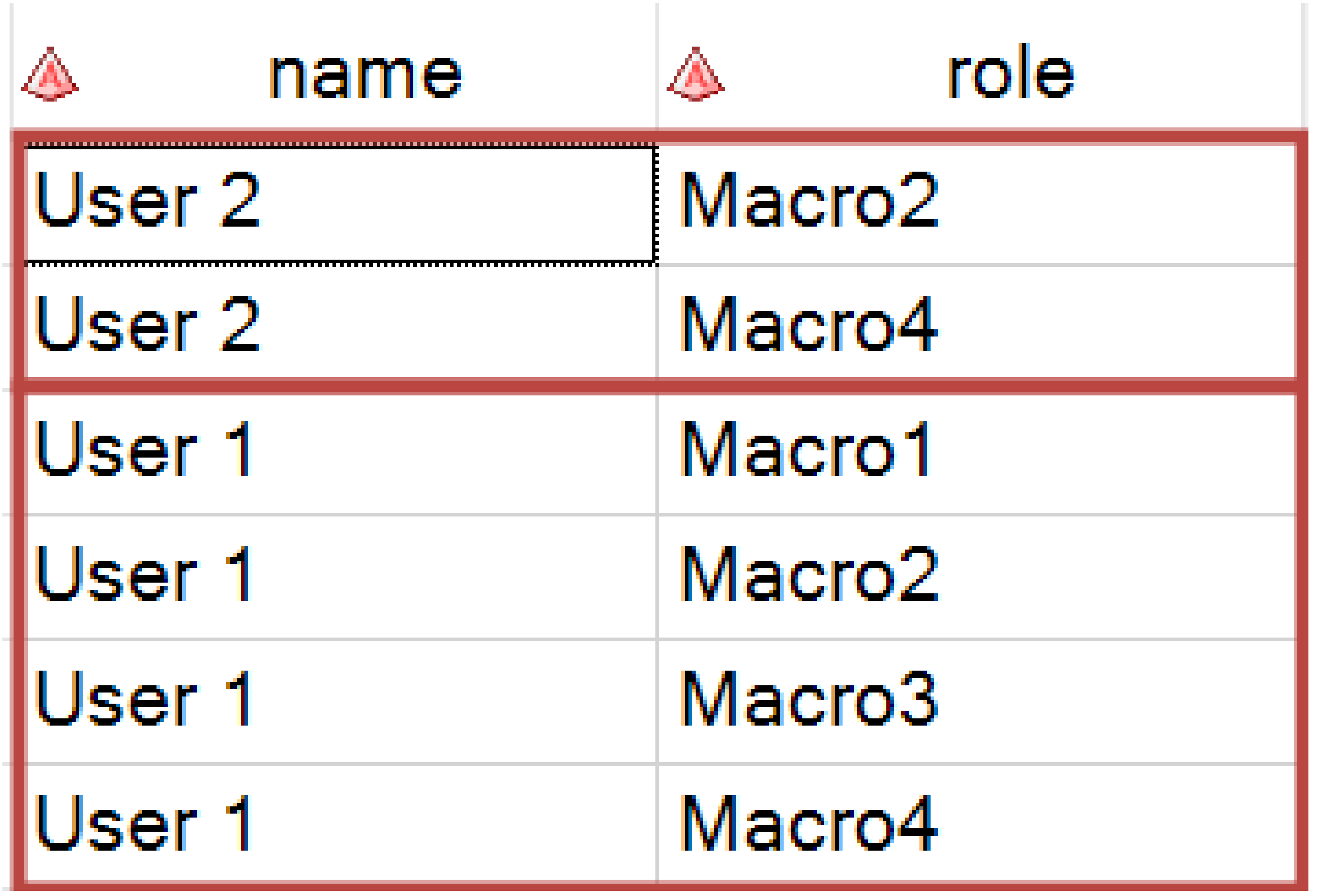


Querying Metadata

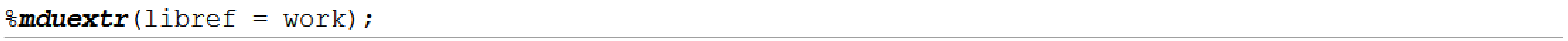
Querying SAS Metadata can be quite complicated, but SAS provides several pre-installed macros to complete the hard work. %MDUEXTR() is the one used for retrieving users, groups, and roles.

This macro pulls back the entire metadata for the given server. This needs to be queried down, via the '&_clientuserid' macro variable. The query below will pull back all of the roles associated with a user, whether assigned directly or indirectly – through a group.

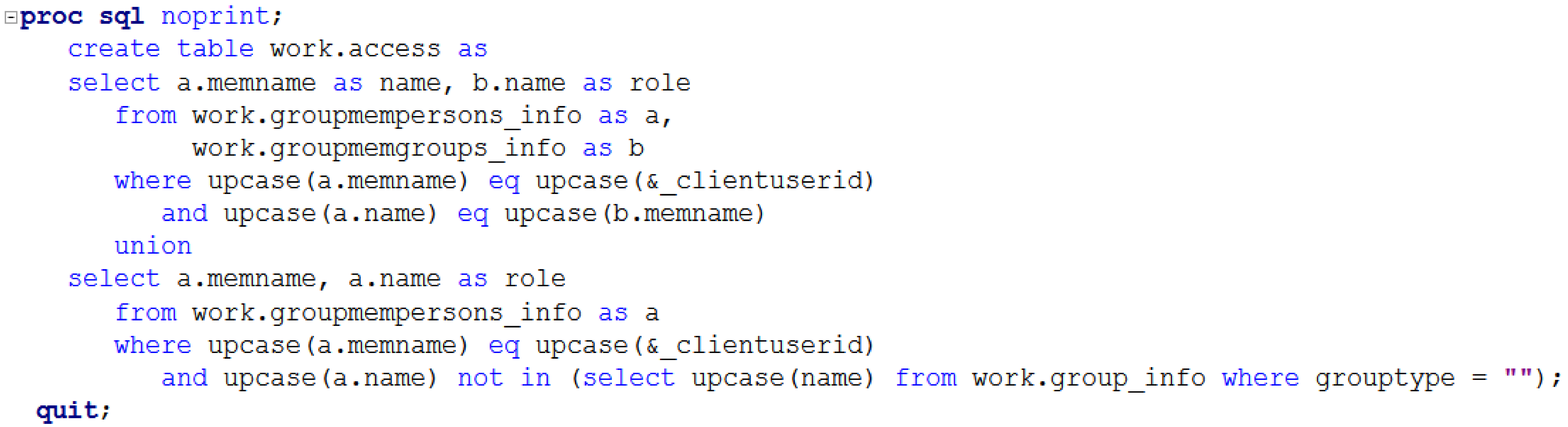
With this data the user can now verify access prior to macro executions.



name	role
User 2	Macro2
User 2	Macro4
User 1	Macro1
User 1	Macro2
User 1	Macro3
User 1	Macro4



```
%mduextr(libref = work);
```

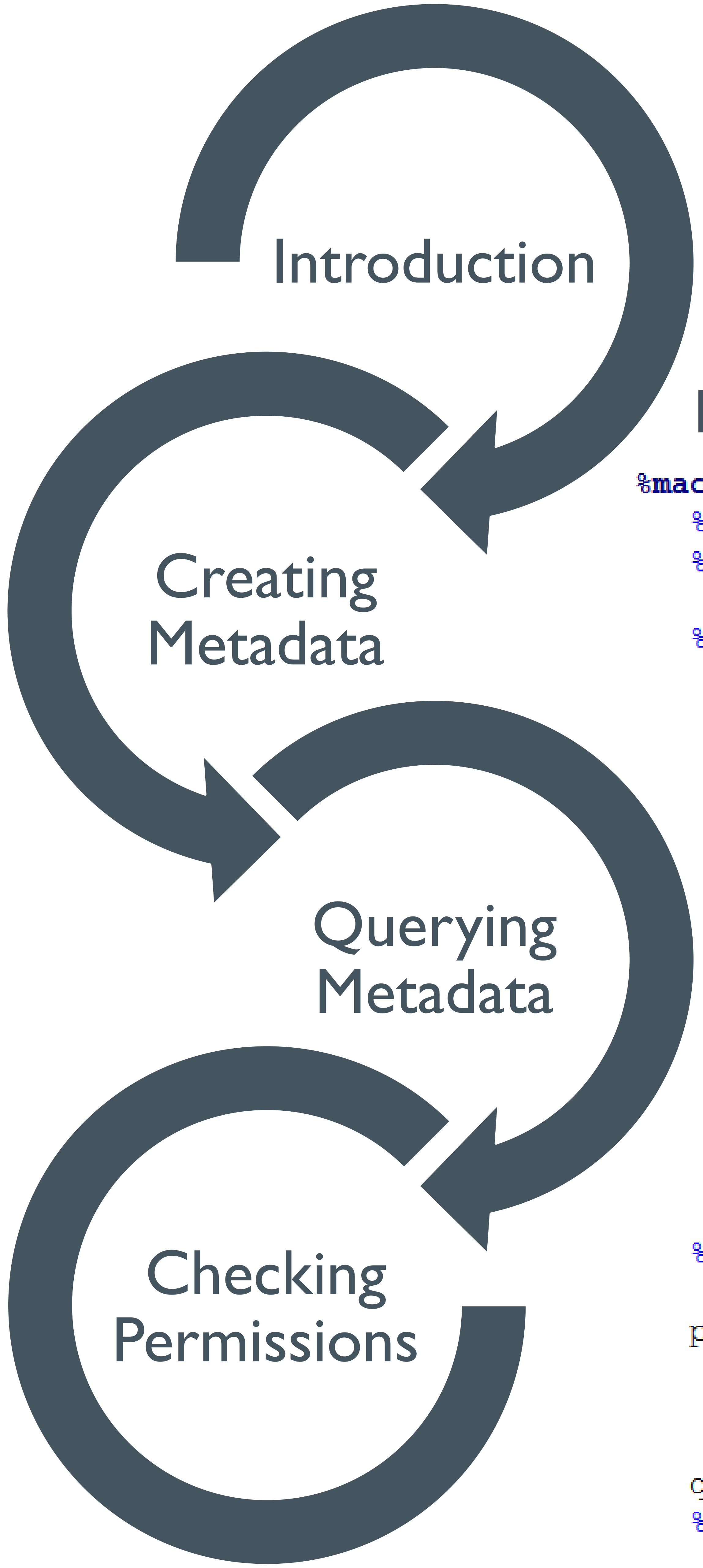


```
proc sql noprint;
   create table work.access as
   select a.memname as name, b.name as role
      from work.groupmempersons_info as a,
           work.groupmemgroups_info as b
      where upcase(a.memname) eq upcase(&_clientuserid)
         and upcase(a.name) eq upcase(b.memname)
      union
   select a.memname, a.name as role
      from work.groupmempersons_info as a
      where upcase(a.memname) eq upcase(&_clientuserid)
         and upcase(a.name) not in (select upcase(name) from work.group_info where grouptype = "");
 quit;
```

Improving Security Through Metadata Objects: Permission-Driven Macros

Andrew Gannon

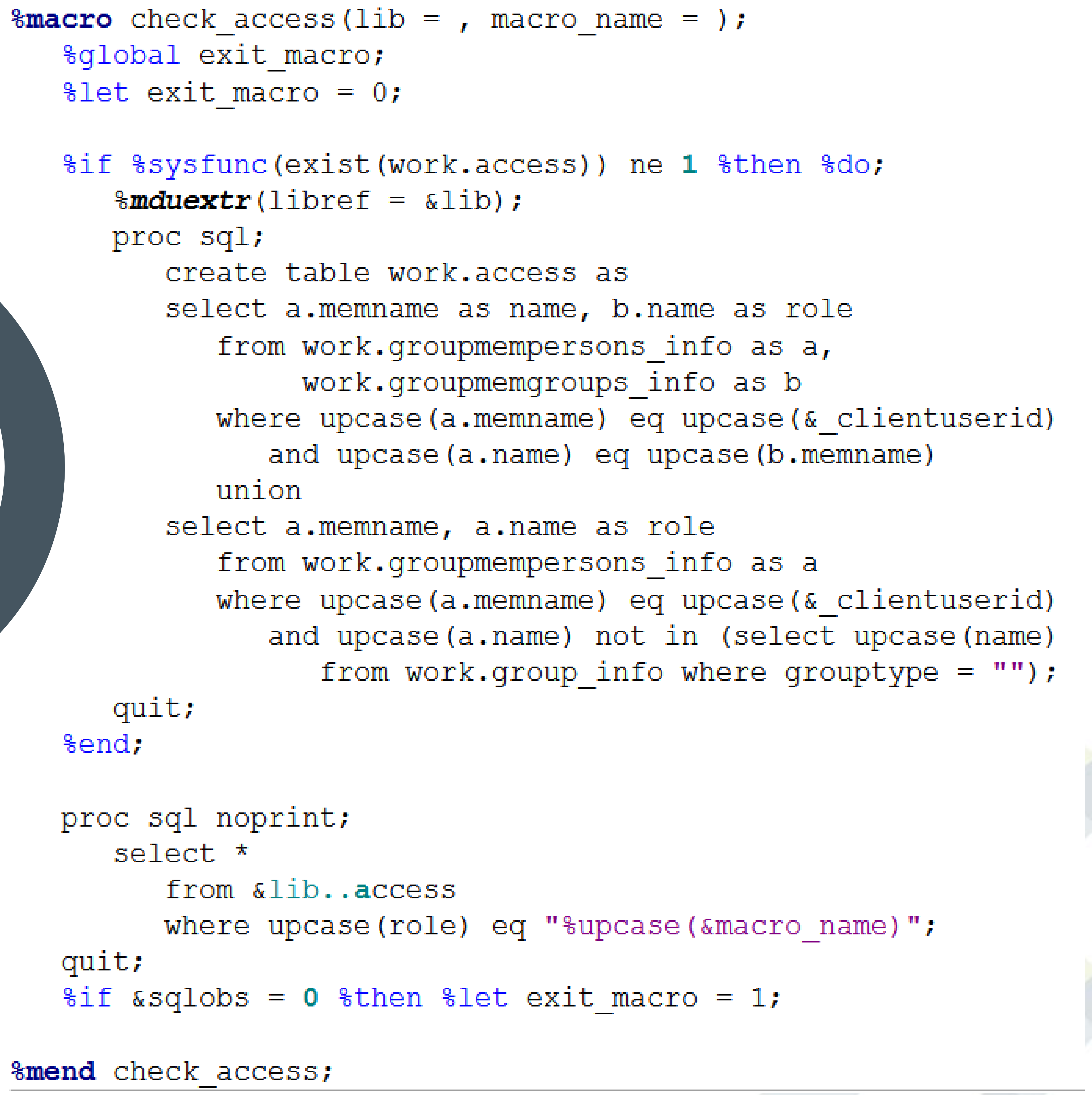
The Financial Risk Group, Cary, NC



Checking Macro Permissions

A macro is created to easily check permissions for other macros. In this macro, we call the code to query the metadata and then compare it to the name of the macro that is being called. If they do not match then the execution is halted.

Macro to Check Access



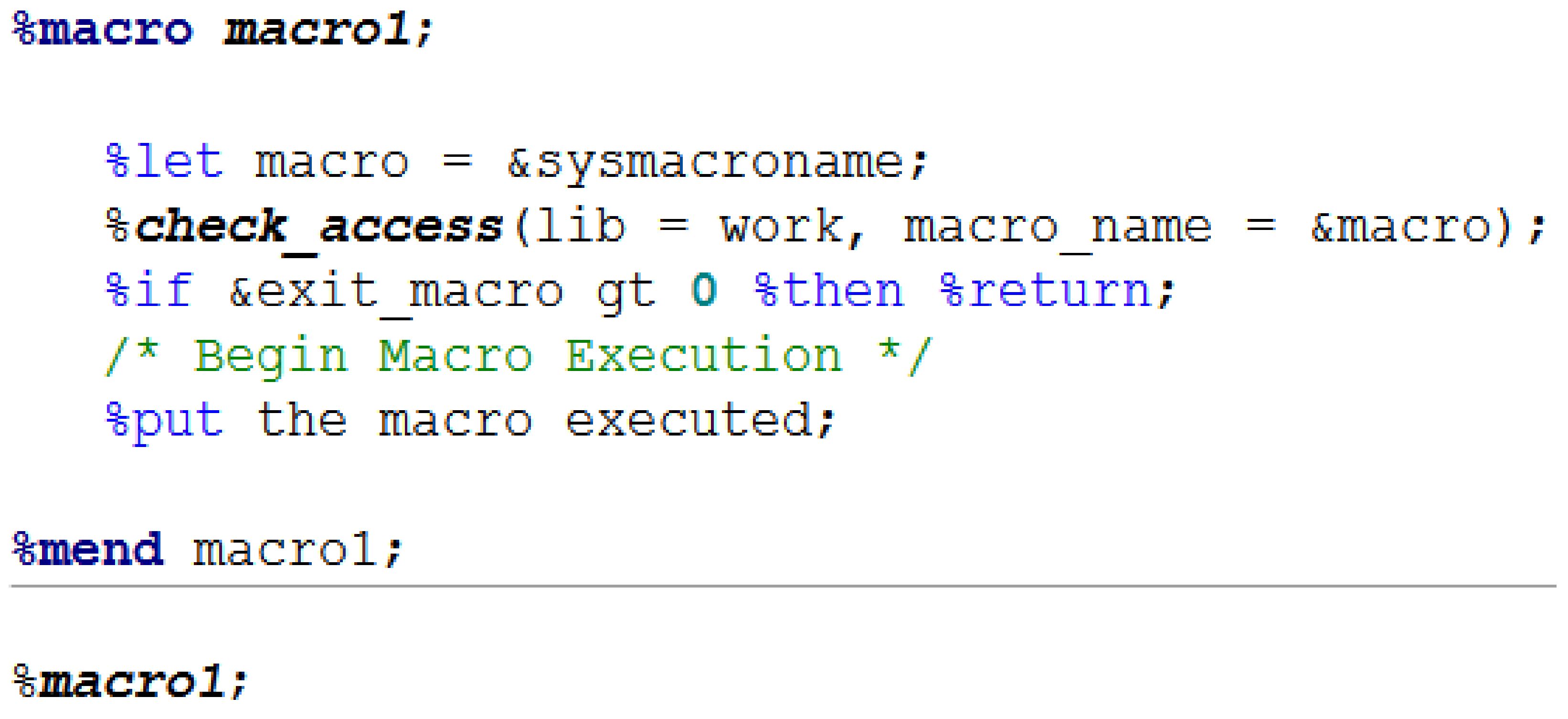
```
%macro check_access(lib = , macro_name = );
   %global exit_macro;
   %let exit_macro = 0;

   %if %sysfunc(exist(work.access)) ne 1 %then %do;
      %mduextr(libref = &lib);
      proc sql;
         create table work.access as
         select a.memname as name, b.name as role
            from work.groupmempersons_info as a,
                 work.groupmemgroups_info as b
            where upcase(a.memname) eq upcase(&_clientuserid)
               and upcase(a.name) eq upcase(b.memname)
            union
         select a.memname, a.name as role
            from work.groupmempersons_info as a
            where upcase(a.memname) eq upcase(&_clientuserid)
               and upcase(a.name) not in (select upcase(name)
                  from work.group_info where grouptype = "");
      quit;
   %end;

   proc sql noprint;
      select *
         from &lib..access
         where upcase(role) eq "%upcase(&macro_name)";
   quit;
   %if &sqlobs = 0 %then %let exit_macro = 1;

%mend check_access;
```

Application of Permissions



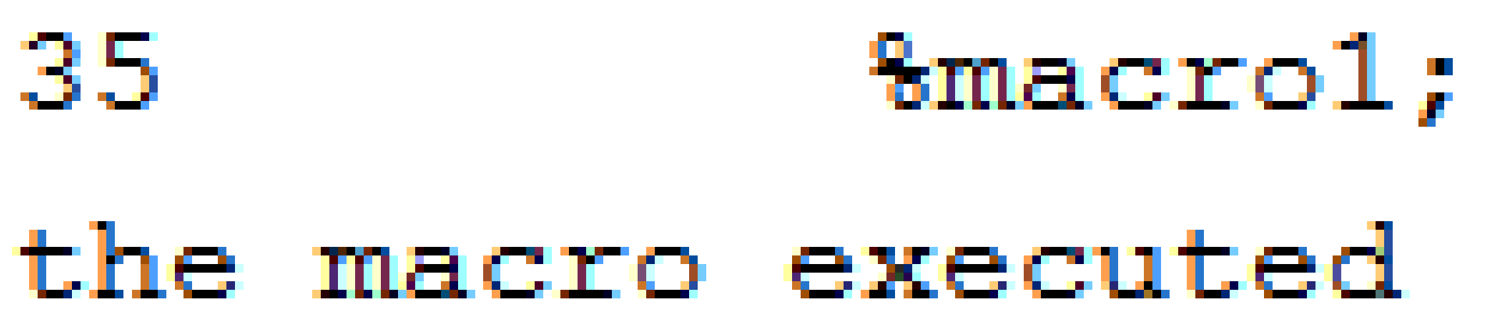
```
%macro macro1;

   %let macro = &sysmacroname;
   %check_access(lib = work, macro_name = &macro);
   %if &exit_macro gt 0 %then %return;
   /* Begin Macro Execution */
   %put the macro executed;

%mend macro1;

%macro1;
```

Application of Permissions



```
35          %macro1;
the macro executed
```

Conclusion

Utilizing metadata and simple SAS functionality to implement permissions on the macro level is an easy and effective way to support the audit function. Whether it’s preventing users from updating tables or from misusing computing power, this method can do the trick.

Paper 3025-2019

Improving Security Through Metadata Objects: Permission-Driven Macros

Andrew Gannon, The Financial Risk Group, Cary NC

ABSTRACT

Users can wreak havoc on systems in many different ways – from excessive memory and cpu usage to updating tables with bad data missing data. There are several ways to prevent individuals from doing this – namely by limiting users' access to servers and data. There is a better way to manage permissions for users that is not server or environment specific. By creating a framework of metadata groups and roles, macro specific permissions can be applied at the individual and group levels. This paper seeks to explain and demonstrate how to apply macro level permissions to users via roles and groups in Metadata.

SAS METADATA

Most, if not all, SAS users are defined in some way as users in SAS Metadata. That means that a SAS Administator created a metadata object for the user that allows them to access and use the differnet SAS products. Typically, products that users have available have associated metadata role objects – which manage the availability of an application or feature. Most SAS Products have pre-defined roles for different products. Collections of roles can be created as metadata group objects. Groups are collections of roles and are assigned to users. Groups are transitive, if a users is assigned to a group, then the user is inherently assigned to all roles associated with groups. Figure 1 illustrates the different metadata objects and how they are linnked. The 'user' is assigned to individual roles as well as a collecection of roles (a group). Figure 2 illustrates how the users is associated with all the roles and objects from figure 1.

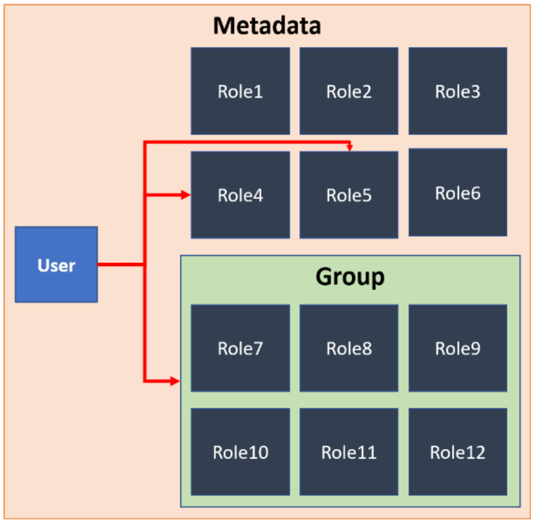


Figure 1

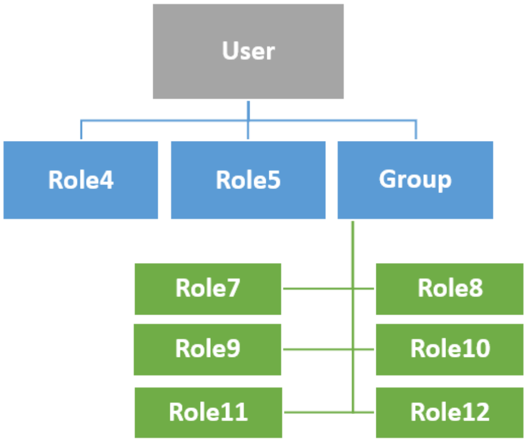


Figure 2

Creating Permissions Metadata

Starting with a good framework is the best way to develop new roles and groups in metadata. Think of all of the possible roles / permissions and then create roles for those permissions. Naming techniques aid in this part. For instance if all the permissions apply to one server or application, have the roles name start with the name of the server or the name of the applicaton. If there is a server names 'DEV' then all permission names would start as *Dev:* …. this would then allow for easy querying and readability of the metadata. Next, develop a framework of users and the roles that they should have. If there are similar paterns then consider building some groups – this allows users to have a singular group assignment rather than manually assigning several roles. Again – naming conventions help with groups as well. Once the roles and groups are created, they can be assigned to users as necessary. For the purpose of this paper we assume all roles and groups start with 'DEV:' followed by the exact name of the macro that it will be applied to. If the macro is called *update_table* then the metadata role would be called 'DEV: UPDATE_TABLE'. These naming conventions will help when determining if a user has access to particular macros. Note that all of these steps involving metadat should be handled by the SAS Administrator responsible for the metadata repository.

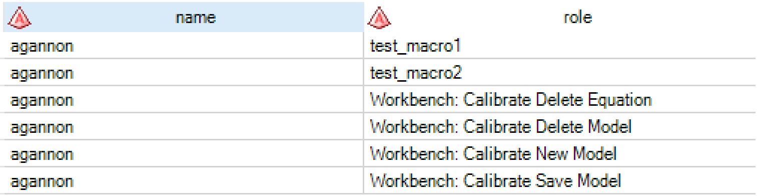
Querying Metdata

Metadata can be pulled down into SAS Datasets through both querying and built-in SAS functions. This paper will not go in-depth on how to use or alter the metadata querying, but rather will explain exactly what to do to get the users, roles, and groups from the metadata and how to use those to permission macros.

SAS metadata is built on an XML Schema and the easiest way to query is through the PROC METADATA procedure. This procedure is like PROC HTTP in that the user specifies an infile and an outfile. The infile is an XML file that contains the querying information. The procedure queries the metadata using the provided a XML-map and returns an XML dataset with the results. From here, the XML LIBNAME is the most efficient way to convert the XML to a usable SAS Dataset. The user can create the process on their own, or they can use a built in SASFoundation macro **%mduextr(libref =)** which pulls down several datasets from metadata including a list of roles, groups, and users. These datasets can easily be joined to generate a list of all roles associated with a user. Utilizing this built in macro allows the user to query the output based on the current user – via &_clientuserid for Enterprise Guide or &_secureusername for a stored process. Figures 3 and 4 show the query used (after calling %mduextr). Utilizing the &_clientuserid (in Enterprise Guide) the user can pull all the available metadata groups and roles associated with them.

```
proc sql;
   create table work.access as
   select a.memname as name, b.name as role
      from work.groupmempersons_info as a,
           work.groupmemgroups_info as b
      where upcase(a.memname) eq upcase(&_clientuserid)
         and upcase(a.name) eq upcase(b.memname);
quit;
```

Figure 3



name	role
agannon	test_macro1
agannon	test_macro2
agannon	Workbench: Calibrate Delete Equation
agannon	Workbench: Calibrate Delete Model
agannon	Workbench: Calibrate New Model
agannon	Workbench: Calibrate Save Model

Figure 4

ADDING PERMISSIONS TO MACROS

Now that a list of roles can be pulled for a user, this list can be used to guide permission-based security at the SAS macro level. The newly created roles and groups don't have any effect as is – they must be applied somehow. This is a simple concept where a permission check is run at the beginning of the macro. For further efficiency we encapsulate this code into its own macro that can simply be run at the beginning of each macro. The goal is to examine the name of the currently executing macro – then determine if the user has the availability to execute this macro by checking associated roles. Once that is determined the macro will either execute or exit.

The Permissions Macro

The permissions macro can be developed one of two ways: (1) it can query the metadata for roles on each call, or (2) it can query metadata on first call then use a saved table on all other calls in the same session. This paper will discuss both methods. The first method is straight forward. The macro will query

down on the current user's groups and then on all roles. It will check that the user has access to current macro – and then execute the macro or exit. The name of the currently executing macro can be found by the automatic SAS macro variable &sysmacroname. This is available throughout the execution of a macro. This is why naming metadata roles based on macro name is important – now the macro name can be compared against metadata – if the names match up then the user has permission to execute.
The second method of saving a table off and using that table through the session to determine permission is a better option if there is a good place to store the table where it won't be deleted or altered throughout the session.
The *check_access()* macro below pulls in the required groups and roles and checks if the passed macro name is in the list of available macros the user has access to.

```
%macro check_access(lib = , macro_name = );
   %global exit_macro;
   %let exit_macro = 0;

   %if %sysfunc(exist(work.access)) ne 1 %then %do;
      %mduextr(libref = &lib);
      proc sql noprint;
         create table work.access as
         select a.memname as name, b.name as role
            from work.groupmempersons_info as a,
                 work.groupmemgroups_info as b
            where upcase(a.memname) eq upcase(&_clientuserid)
               and upcase(a.name) eq upcase(b.memname);
       quit;
   %end;

   proc sql noprint;
      select *
         from &lib..access
         where upcase(role) eq "%upcase(&macro_name)";
   quit;
   %if &sqlobs = 0 %then %let exit_macro = 1;

%mend check_access;
```

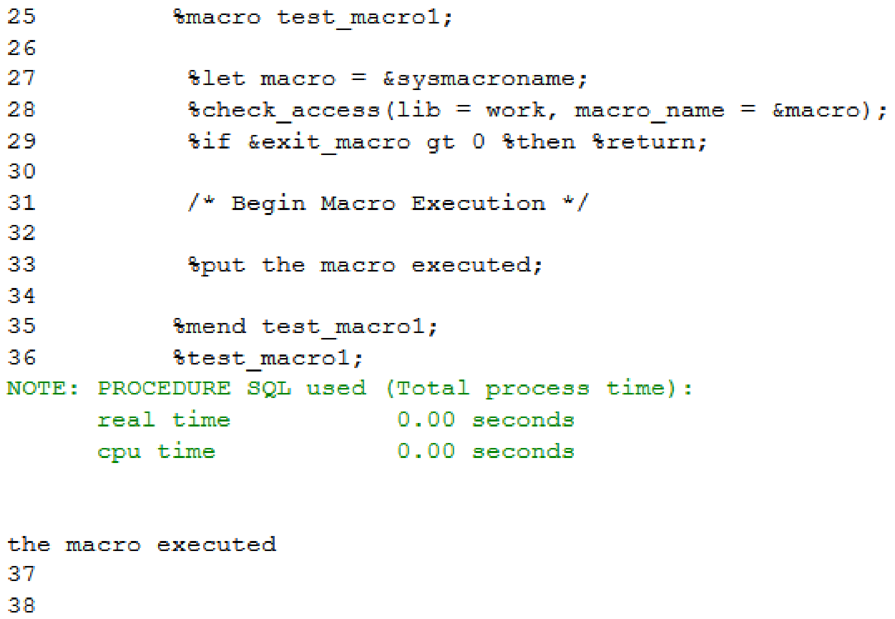
The *check_access()* macro can then be called at the beginning of macros. The user first initializes the current macro name to pass, then checks the value of the output global macro variable *exit_macro* to determine if access is available. Below is an example.

```
%macro test_macro1;

   %let macro = &sysmacroname;
   %check_access(lib = work, macro_name = &macro);
   %if &exit_macro gt 0 %then %return;
   /* Begin Macro Execution */
   %put the macro executed;

%mend test_macro1;
%test_macro1;
```

The following macro executes and prints the code at the bottom. If the name is changed to that of something not in the list of available metadata macros, then the program would have exited. Below is the printout of the log from executing this macro.



```
25         %macro test_macro1;
26
27          %let macro = &sysmacroname;
28          %check_access(lib = work, macro_name = &macro);
29          %if &exit_macro gt 0 %then %return;
30
31          /* Begin Macro Execution */
32
33          %put the macro executed;
34
35         %mend test_macro1;
36         %test_macro1;
NOTE: PROCEDURE SQL used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds


the macro executed
37
38
```

Figure 5

CONCLUSION

Permissioning at the macro level allows businesses and organizations the ability to closely monitor activities. Knowing users that are and are not allowed to execute macros greatly increases security and helps with auditing. It is important to build these permissions as it allows for the accurate updates of tables and proper use cpu and memory.

CONTACT INFORMATION

If you have any comments or concerns, please feel free to reach out at any time. Please contact at:

Andrew Gannon

The Financial Risk Group, Inc.

+1 (919) 439-3819

Andrew.Gannon@frgrisk.com

www.frgrisk.com