Paper 321-2009

Using SAS® 9.2 Metadata Security Reporting and Auditing Features

Forrest Boozer and Diane Hatcher, SAS Institute Inc., Cary NC

ABSTRACT

Generating reports of user access rights or knowing who has changed security on a resource are critical aspects of a secure computing environment. In SAS® 9.1.3, the SAS® Metadata Server played a central role in the security of SAS resources. However, there were no tools or reports provided to indicate which users or groups had access to a particular resource, or who had changed the security on it.

This paper explains the new SAS® 9.2 security auditing capabilities built into the SAS Metadata Server and examines how this information might be used. In addition to auditing, security reporting macros are provided. We examine how to use these macros to extract the security settings for metadata objects into data sets that can be used to produce reports.

INTRODUCTION

The SAS 9.2 Platform release brings major changes that provide administrators knowledge of the state of their environment, and knowledge of the changes that are being made. Those improvements have come as a result of two significant efforts. One is to leverage a new logging facility in the SAS Metadata Server by creating audit records for changes that affect the security of the environment. These records include information about the date, time, who, and type of change to an object. It does not tell you exactly what changed, but it notifies you that a change was made, and by whom and when. In some cases, additional logging can be enabled so that details of the state of the object after the change are logged.

The second area of improvement is the ability to retrieve effective permission settings from an administrative API in the SAS Metadata Server. This API allows the **Authorization** tab to accurately render the exact authorization decisions that will be made for a particular object. This same API is available to all users through functions in the DATA step. This means that reporting is now possible on authorization settings. Macros are provided in SAS 9.2 that use the DATA step functions to access the effective permissions on objects in the repository. These macros harvest the permission information and place it in SAS data sets. Once the information is available as a data source, you can use the powerful reporting capabilities of SAS to look at the state of security on a broad set of objects.

This paper explores both the auditing and reporting capabilities in SAS 9.2. We discuss the principles that shaped these features and the implementation details that you need to understand what is available and how best to use it. Several examples are provided in each area to demonstrate their use in solving administration and reporting issues.

AUDITING

In SAS 9.2, auditing and logging capabilities have been improved for all servers. At the heart of this improvement is a standardized logging facility that provides a wide variety of configuration options. This logging facility offers many great new features, including the following:

- Log events are categorized hierarchically and can be configured broadly at the top of the hierarchy or at a very fine-grained level.
- A variety of log appenders are available for configuration. These appenders allow events to be sent not only to log files, but also to operating system facilities and client applications.
- Log events can be configured to go to multiple and different appenders, based on the logging configuration file and filters that can be applied to the appenders.
- The format of the log message itself can be customized to ensure that information that is relevant to the event and its relationship to other events is included in the log.

For information about how to configure logging for the SAS Metadata Server, see Chapter 8, “Administering Logging for SAS Servers,” in the *SAS 9.2 Intelligence Platform: System Administration Guide*. In-depth information about the logging facility can be found in *SAS 9.2 Logging: Configuration and Programming Reference*.

OVERVIEW OF SECURITY AUDITING IN 9.2 METADATA

The 9.2 SAS Metadata Server takes advantage of this new logging facility by providing metadata-specific messages in six different categories. Those categories are listed using the hierarchal name by which they are known in the logging system.

App.Meta—The default category for general metadata informational, warning, and error messages.

Audit.Meta.Security—All events that are triggered by changes to metadata that could affect or change authorization decisions for objects in the server. There are several subcategories of messages that are in this area, each of which is discussed later.

App.Meta.IO—Messages from low-level I/O routines.

App.Meta.Mgmt—Logs events related to the operation and management of the server.

Perf.Meta—Logs operational statistics about SAS® Open Metadata Interface method calls. This information previously was available through the SAS® Open Metadata Architecture application response measurement (ARM) logging system.

App.Meta.CM—Logs event related to change management. (Checking objects into and out of project repositories.)

Additional categories of events are available from the integrated object model (IOM) infrastructure that is used to implement SAS servers. These additional logging categories can be configured to track the majority of server interactions with client applications.

The auditing aspect of this paper is focused on the events associated with the Audit.Meta.Security category. When developing the messages and events that would be emitted by this category, our goal was to log changes being made to the security infrastructure of the metadata. This included changes that could affect a user's ability to access or change metadata or the data it represents. Emphasis was placed not so much on the details of the change itself, but on identifying the type of change being made, the objects being affected, and the user making the change.

By using the logging facility, these events can be sent to a secure logger. (A secure logger prevents a user or administrator from removing logged events.) Logging these events has the following advantages:

- Deterrence—If users or administrators know that their activities are being logged, then they will be less likely to take advantage of their privileges.
- Monitoring—Allows you to monitor security activity in the server.
- Diagnostic—When access to an object fails because of an authorization change, you can determine who made the change and work on a resolution with them. This is handy when a user or program changes a user ID or password and suddenly access to a resource is lost. The event in the log allows you to identify which logins were changed and who changed them.

DEFAULT SETTINGS FOR METADATA SECURIYT AUDITING IN 9.2

Regardless of whether you migrated from 9.1.3 to 9.2, or you are building a new 9.2 SAS deployment, the auditing settings that are put in place for the 9.2 SAS Metadata Server are the same. All messages are written to the SAS Metadata Server log files, whose default location is the **…\lev1\sasmeta\metadataserver\logs** directory of the SAS configuration. By default, the entries are written to a RollingFileAppender, which creates a new log file each day. Old log files are left in place for reference. To make the log files easy to identify, they include the date the log was started and the process ID of the SAS Metadata Server.

READING LOG ENTRIES

Each logged event in the server has an associated message. When the event occurs, the message is sent to the logging facility so that the message can be placed in the appropriate log. When the message is written to the log, the logging facility applies a conversion pattern to the event. The conversion pattern defines a format for the log entry of the message that can include additional information about the time and origin of the event. The conversion pattern is configurable, and there are many items that can be included. The following is an example of a log entry generated when a new access control template (ACT) was added to the SAS Metadata Server:

```
2009-02-12T16:38:04,646 INFO  [00001672] 4:sasjfb@CARYNT - Added AccessControlTemplate
Name=SAS Administrator Settings, ObjId=A5TB234I.AM000001.
```

The log entry appears as a single line in the log file. The message from the event is located at the end of the entry: **Added AccessControlTemplate Name=SAS Administrator Settings, ObjId=A5TB234I.AM000001**. Prepended to that message is identifying information that was defined by the default conversion pattern configured for a 9.2 SAS Metadata Server. The information prepended by the conversion pattern is explained as follows:

2009-02-12T16:38:04,646 INFO [00001672] 4:sasjfb@CARYNT This is the timestamp when the event occurred. It has a format of *yyyy-mm-dd*, representing the date, followed by the static character **T** and the time in a 24-hour format of *hh:mm:ss,SSS*. (The *,SSS* represents milliseconds.) Because of the order of the numeric representation of the date and time, this field can be used in a character sort to order the events based on the timestamp.

2009-02-12T16:38:04,646 INFO [00001672] 4:sasjfb@CARYNT This field is fixed five-characters wide and represents the level of the event (discussed in "Logged Events").

2009-02-12T16:38:04,646 INFO [00001672] 4:sasjfb@CARYNT This represents the thread in the server that generated the event. A single request to the SAS Metadata Server can generate multiple events. The events for that request all have the same thread identifier. When multiple requests are being processed simultaneously by the server, it is possible for events for the different requests to be interlaced in the log. This provides a way to pull the related events together.

2009-02-12T16:38:04,646 INFO [00001672] 4:sasjfb@CARYNT This is a Mapped Diagnostic Context. Basically, it is a numeric identifier that can be used to identify the client connection that submitted the request that generated the event. This identifier is also found in new client connection events. The message for the new connection event also includes the IP address and port of the client application. This can be very useful in determining the source of a rogue request.

2009-02-12T16:38:04,646 INFO [00001672] 4:sasjfb@CARYNT This is the client user ID that is associated with the thread that generated the event.

LOGGED EVENTS

The logging facility associates a level with each message in a category. The levels allow a logging configuration to be tuned to an appropriate level of detail. The levels, from most detailed to least detailed, are TRACE, DEBUG, INFO, WARN, ERROR, and FATAL. For the App.*xxx* and Audit.*xxx* categories, events from the INFO level are logged. These settings are detailed enough to see basic operations of the server, client connections, and administrative activities that affect metadata access and security.

Within Audit.Meta.Security are three subcategories, each with multiple messages. These subcategories track different aspects of security infrastructure change. The messages generally have a common form. They indicate that a particular item was added, updated, or removed, or that an attempt to perform one of those actions was made by a user who is not authorized. The subcategories of Audit.Meta.Security and examples of their messages are as follows:

Audit.Meta.Security.UserAdm

Pertains to user administration events, specifically the creation of identities (users, groups, roles), logins (user IDs and passwords), and authentication domains. Here are some examples of these messages:

Added IdentityType=IdentityGroup Name=SASUSERS, ObjId=A5TB234I.A3000001.
This message was generated when the SASUSERS group was added to the SAS Metadata Server during the initialization of the Foundation repository. Any user, group, or role that is created generates a message in this form. Additionally, if an identity is changed or removed (deleted), similar messages are generated.

Added Login with UserId=conf67\tina, ObjId=A5TB234I.AQ000008, AuthDomain=DefaultAuth to IdentityType=Person Name=Tina, ObjId=A5TB234I.AN000008.
When an account (for example, user ID and optional password) is added to the **Accounts** tab of a user or group in SAS® Management Console or SAS® Personal Login Manager, a Login object is added to the Person or IdentityGroup object in the metadata. The message is an example of adding a Login object with a user ID of **conf67\tina** to the user (Person object) with the name Tina. Remember that the NAME attribute of an Identity object is unique within the server for objects of the same type. So, this should be enough to identify the user or group to which the Login object was added. The metadata type and object IDs are provided if more specific information is needed.

Added Authentication Domain Name=DefaultAuth, ObjId=A5TB234I.AP000001.

Authentication domains are used to associate logins to the servers where they are valid. Changes that impact authentication domains generate messages in the form above.

Changed Internal Login ObjId=A5TB234I.AO000001 for Person Name=sasadm, ObjId=A5TB234I.AN000001.

Internal logins are created by the SAS Metadata Server when an internal account is added to a user in SAS Management Console. The message was generated by the initialization of a Foundation repository when an internal password was modified on the SAS Administrator account. (Keep in mind that SAS Administrator is the display name for the user. The actual name attribute for the user is SASADM.) Any change to the internal account, such as changes to the expiration date, lockout policy, and password reuse generates a change event like the message above.

Audit.Meta.Security.GrpAdm

Pertains to group and role membership administration events, specifically any change to the membership of a group or role generates an event. Note that changes to the name or other attributes of a group do not generate messages in this category. Those changes generate events in the UserAdm category. Here are two examples:

Added Member IdentityType=Person Name=sastrust, ObjId=A5TB234I.AN000002 to IdentityGroup Name=SAS System Services, ObjId=A5TB234I.A3000007.

This message provides an example of a user (NAME=SASTRUST) being added to the group SAS System Services.

Added Member IdentityType=IdentityGroup Name=SASAdministrators, ObjId=A5TB234I.A3000003 to Role Name=META: User and Group Administrators Role, ObjId=A5TB234I.A3000005.

This message shows the addition of a group (NAME=SASADMINISTRATORS) being added to the role META: User and Group Administrators Role.

Audit.Meta.Security.AccCtrlAdm

Pertains to access control administration events. These events usually involve creating and manipulating access controls on objects and ACT definitions. Messages in this subcategory include the following:

Added Permission Name=ReadMetadata, Type=GRANT, ObjId=A5TB234I.A2000001, Repository=Foundation.

This message is generated during initialization of Foundation repositories when permission objects are added to the system. Permission objects should not be added, updated, or deleted by your applications. The initialization process of the Foundation repository should add the appropriate set of permissions. Those permissions will be used by the authorization system. Other additions, changes, or deletions of permission objects should not occur.

Access Control change on ObjectType=Tree, Name=System, ObjId=A5TB234I.AJ000004.

When a change is made to the access controls that protect an object, an event is generated with a message in the form above. Access control changes include granting, denying, and removing permission settings for a user. It also includes applying or removing an ACT on an object. This message tells us that a user changed the access controls that protect the tree named System. The folders that you see when browsing metadata are implemented using Tree objects. This message was generated during the initialization of the Foundation repository when the System folder is being created and access controls are being defined for it.

Added AccessControlTemplate Name=SAS Administrator Settings, ObjId=A5TB234I.AM000001.

ACTs contain a set of permission settings. These templates can be set on the access controls of multiple objects, conveying to those objects the setting contained in the ACT. The message was generated when the SAS Administrator Settings ACT was created.

Access Control definition change on AccessControlTemplate Name=Default ACT, ObjId=A5TB234I.AM000003.

This message is generated when the permission pattern stored inside an ACT is changed. This permission pattern is the settings that are shown on the **Permission Pattern** tab of the ACT Properties dialog box. Keep in mind that an ACT can also be the subject of an access control change event. In this case, the

permissions that control access to the ACT are being changed, not the permission pattern stored in the ACT.

The messages above are defined for the INFO level of logging. But, this category has additional logging that can be enabled at the TRACE level. When the TRACE level is enabled, each time permissions are changed on an object or an ACT definition is changed, the resulting effective permissions are written to the log. The following is an example that was generated when a change was made to the default ACT definition:

```
2009-02-19T13:22:59,787 TRACE [00008047] 16:tina@conf67 - Trace log showing
     permissions defined for : OMSOBJ:AccessControlTemplate/A5TB234I.AM000003.
    SASUSERS IdentityGroup ReadMetadata=EG|ND, WriteMetadata=EG|ND,
       WriteMemberMetadata=ND, CheckInMetadata=EG|ND, Read=ND, Write=ND,
       Administer=ND, Create=ND, Delete=ND, Execute=ND, Create Table=ND,
       Drop Table=ND, Alter Table=ND, Select=ND, Insert=ND, Update=ND,
       References=ND
    PUBLIC IdentityGroup ReadMetadata=ED, WriteMetadata=ED,
       WriteMemberMetadata=ED, CheckInMetadata=ED, Read=ED, Write=ED,
       Administer=ED, Create=ED, Delete=ED, Execute=ED, Create Table=ED,
       Drop Table=ED, Alter Table=ED, Select=ED, Insert=ED, Update=ED,
       References=ED
    SASAdministrators IdentityGroup ReadMetadata=EG|NG, WriteMetadata=EG|NG,
       WriteMemberMetadata=ND, CheckInMetadata=EG|NG, Read=ND, Write=ND,
       Administer=EG|ND, Create=ND, Delete=ND, Execute=ND, Create Table=ND,
       Drop Table=ND, Alter Table=ND, Select=ND, Insert=ND, Update=ND,
       References=ND
    SAS System Services IdentityGroup ReadMetadata=EG|NG, WriteMetadata=EG|NG,
       WriteMemberMetadata=ND, CheckInMetadata=NG, Read=ND, Write=ND,
       Administer=ND, Create=ND, Delete=ND, Execute=ND, Create Table=ND,
       Drop Table=ND, Alter Table=ND, Select=ND, Insert=ND, Update=ND,
       References=ND
```

In the SAS Metadata Server log, the message above appears as five separate lines. The first line is the TRACE event message. Following that are four lines, one for each user or group that appears in the default ACT definition. For each identity, a list of permissions is given with an indication of the setting for that user. The values for the settings are as follows:

EG – Explicitly Granted
ED – Explicitly Denied
AG – ACT Grant, applied ACT provided the grant
AD – ACT Deny, applied ACT provided the deny
NG – Indirectly Granted (by group membership or inheritance)
ND – Indirectly Denied (by group membership or inheritance)

While this message does not show you what was changed, it does show you the net result after the change was made.

USING THE AUDIT ENTRIES

The SAS Metadata Server log files contain a potentially large amount of information about the activities that have occurred in the server. While it is tempting to write a SAS DATA step program to extract information, it might not be necessary. If there are specific events you want to be notified of or want to monitor, you can configure additional appenders for the logging configuration and have those events automatically logged in a specific location, in addition to the default metadata log. This is nice because a file containing just the information you are interested in is built as the events occur. There is no need to write programs to scan the logs looking for the information. If issues are discovered in this special log file, you can use that information to find the same event in the metadata log files and discover the context of the event in relationship to other activities in the server. Observing the context is often valuable when trying to diagnose who made a particular change or why a particular change was made.

EXAMPLE—WHO CHANGED SECURITY ON A FOLDER

Generally, audit information is used to determine what has happened in the system and who or what caused it. For this example, assume that a user has called and complained that they can no longer access the stored process samples in the **`\Products\SAS Intelligence Platform\Samples`** metadata folder. The first thing you do is check the authorization settings on the folder and find that the settings have been changed. SASUSERS is now

denied access and only the CONTENT CREATORS group has access. Before changing the permission setting to allow SASUSERS, you should find out who made the change and why. (There might be a reason that you are not aware of.) This can be done by searching the SAS Metadata Server log files for an access control change event on the object. If you know when the change is likely to have occurred, then you can simply search the log file for that day using a text editor. However, if you do not really know when the change occurred, you might find it more useful to use a utility like **grep** on UNIX, or **findstr** on Windows to search all the log files.

For Windows, issue the following command from a command prompt with the current directory in the SAS Metadata Server Logs directory (**…\Lev1\SASMeta\MetadataServer\Logs**):

```
findstr /c:"Access Control change" *.log | findstr /c:"ObjectType=Tree, Name=Samples,"
```

The **findstr** command as specified searches all of the .log files in the current directory, looking for lines that contain the character string "Access Control change." Those lines are then piped using the | operator to another **findstr** command that further filters the output to only those changes to an object with an object type of TREE and a name of a Samples. (It is important to know that folders in the SAS Metadata Server are implemented using TREE metadata objects. The condition for our search used an object type of TREE rather than FOLDER.) The command returned the following output:

```
SASMeta_MetadataServer_2009-02-26_340.log:2009-02-26T22:33:02,931 INFO  [00002337]
12:tina@conf67 - Access Control change on ObjectType=Tree, Name=Samples,
ObjId=A5TB234I.AJ00000J.
```

While the output of the search is tightly packed and awkward to read, it does contain the information you were seeking. The beginning of the output for the line contains the filename of the log file in which the line was found, followed by a colon. After the colon, the log entry for the event is printed. Stripping away the filename leaves you with the following:

```
2009-02-26T22:33:02,931 INFO  [00002337] 12:tina@conf67 - Access Control change on
ObjectType=Tree, Name=Samples, ObjId=A5TB234I.AJ00000J.
```

Using the information discussed previously, you can see from the log that the user who authenticated with the user ID **tina@conf67** made the change. With this information, you can contact the user who owns that user ID and determine why these changes were made. You might discover that other changes were made for the same reason. To identify those changes, you might want to change your search criteria to look for all of the access control changes made by **tina@conf67** on a particular day. Because all the authorization changes generate an audit entry in the log, you should be able to locate all of them.

EXAMPLE—LOCATING THE ORIGIN IP ADDRESS OF A CHANGE

Continuing with the previous example, if the user **tina@conf67** insists that she did not make the change, then you will probably want to investigate a little deeper. Is she mistaken about having made the change, or has her user ID and password been compromised? It might be helpful to know the client IP address from where the change originated. If the IP address maps to the PC belonging to **tina@conf67**, then there is a good chance that she simply forgot making the change. If it does not, then it could mean that an unauthorized individual is using her account. Knowing the IP address might not give you the answer, but, at the very least, you will have an important piece of the puzzle in figuring it out.

To determine the IP address, you need to examine the event log entry for the access control change in the previous example:

```
2009-02-26T22:33:02,931 INFO  [00002337] 12:tina@conf67 - Access Control change on
ObjectType=Tree, Name=Samples, ObjId=A5TB234I.AJ00000J.
```

From this entry, you can trace the mapped diagnostic context (highlighted) from the change event back to the new client connection event. Usually the connection occurs on the same day as the change event. So, rather than using the **findstr** (or **grep**) command, just use a text editor to open the log file in which the change event was logged, and then search for the character string `New client connection (12) accepted`. In our example, that brings us to the log entry:

```
2009-02-26T22:32:15,314 INFO  [00002053] :tina@conf67 - New client connection (12)
accepted from server port 8561 for user tina@conf67.  Encryption level is Credentials
using encryption algorithm SASPROPRIETARY.  Peer IP address and port are
[172.16.17.178]:2312.
```

From this information, we can find the time the connection was made and the IP address and port on the client machine. With that information, a network administrator should be able to identify the machine from which the request originated. (Hint: **`nslookup`** will take an IP address and return the machine name.) If it is a single-user computer, then you have a pretty good lead on who made the change.

ACCESSING THE SAS METADATA SERVER LOG FROM SAS MANAGEMENT CONSOLE

A new feature of the Server Manager plug-in of SAS Management Console is the ability to monitor activity and change server settings directly. After establishing an administrative connection to the server, an administrator can view the events being logged as they occur. This is a great feature because you can observe the activity as it happens from SAS Management Console, without accessing the log files directly on the host machine. An additional feature is the ability to dynamically adjust the logging. You can increase or decrease the logging level of any category without having to stop and restart the SAS Metadata Server. This has great potential when debugging or testing an issue because you can increase logging to capture the exchange only when the test is being run, and then return it to its normal setting. All of this is performed without flooding the log or disturbing other users by having to bring the server down. For more information about this feature, see Chapter 7, “Monitoring the Activity of SAS Servers,” in the *SAS 9.2 Intelligence Platform: System Administration Guide*.

SECURITY REPORTING

Security reporting provides information about the state of security settings. While there are many aspects to security reporting, we are focused on the authorization settings and effective permissions that are set on the objects in the SAS Metadata Server.

The primary objective of security reporting in SAS 9.2 is to provide you with the tools necessary to build data sets containing the security settings for objects in metadata. Once the information is in data set form, you can use SAS to analyze and report on the content. Clearly, reports can be generated showing the current authorization settings on the object. But, you can also compare extractions at different times to determine exactly what has changed. Some changes might cause you to go back and inspect the audit logs to determine who made the change and when.

A series of autocall macros are being provided to create these data sets. These macros take advantage of new DATA step functions that call the same SAS Metadata Server API used by the **Authorization** tab in SAS Management Console to set and obtain effective permission settings.

USING THE MDSECDS MACRO

The primary mechanism for generating the reporting data sets is the %MDSECDS() macro. Given a metadata folder, this macro extracts descriptive information about all of the objects in the folder and all of the effective authorizations applied to those objects. The macro takes the following form:

```
%mdsecds(folder="/", includesubfolders=YES, membertypes="*",
       Memberfilter="", perms="", IdentityTypes="", IdentityNames="",
       Outdata=work.mdsecds );
```

All of the parameters are optional. If none are specified, the macro starts at the root folder of the metadata and extracts information about all folders and their members. If you have a large amount of metadata, this could be rather time-consuming and resource-intensive. So, you might want to target specific subfolders or object types with your extractions. Full documentation for the %MDSECDS() macro can be found in Chapter 10, “Security Report Macros,” of the *SAS 9.2 Intelligence Platform: Security Administration Guide*.

Here are the parameters that you are most likely to use:

Folder—The top-level metadata folder for which member object and authorization information is extracted.

Membertypes—Specifies the public object types that you want reported. By default, all objects are extracted.

Memberfilter—Expression used to filter the members that are extracted. By default, there is no filter.

outdata—The base name of the data sets or views that will be created by the macro itself or the macros it calls. The default is work.mdsecds. Data sets that are created have an underscore and a suffix appended to the base name. Data sets that are created are as follows:

*xxx*_objs—data set containing folder and member information

*xxx*_permsl—permissions in long form, each permission grant or deny is listed on a separate row

*xxx*_permsw—permissions in wide form, the *xxx*_permsl data set is transformed so that each permission is treated as a column with grant or deny as the value

*xxx*_pconds—contains permission conditions for conditional grants

*xxx*_join—the ultimate output of the MDSECDS macro, this view joins the *xxx*_objs data set with permissions in the *xxx*_permsw data set

For example, to generate a data set named permlib.sampperms with authorizations for objects in the **/Products/SAS Intelligence Platform/Samples**, folder, submit the following SAS code:

```
Options metaserver=metaserver.sas.com metaport=8561 metauser="sasadm@saspw"
metapass="{sas002}B38A003D0753BD144FE78A1422CD72AB";

libname permlib "C:\SAS\EBIserver\Lev1\SASMeta\Data\permdata";

%mdsecds(folder="/Products/SAS Intelligence Platform/Samples",
         outdata=permlib.sampperms);

data permlib.sampperms;
   set permlib.sampperms_join;
   run;
```

The macro extracts data sets and provides a view that combines the object data with the permission data.

The resulting data set combines information from the permlib.sampperms_objs and the permlib.sampperms_permsw data sets. The join is performed so that there is one row for each user who would appear on the **Authorization** tab, for each object in the folder. In each row, there are columns that provide unique identifying information about the object and user or group, and the grant or deny status for each permission encountered during the extraction process. An example of the values from a row in the data set is listed in Table 1:

Column Name	Value
ObjId	A5TB234I.AY000002
ObjName	STP Samples
Location	/Products/SAS Intelligence Platform/Samples/
MetadataType	SASLibrary
PublicType	Library
Desc	SAS Stored Process Samples
ParentObjId	A5TB234I.AJ00000J
MetadataCreated	12Feb2009:21:40:16
MetadataUpdated	18Feb2009:16:38:49
ObjUri	omsobj:SASLibrary/A5TB234I.AY000002
Permissions	
IdentityDispname	Content Creators
IdentityName	Content Creators
IdentityType	IdentityGroup
Permission Columns	
ReadMetadata	Granted Indirectly
WriteMetadata	Denied Indirectly
Administer	Denied Indirectly
More Permissions...	

Table 1. Example Record Generated by %MDSECDS()

The column headings should indicate the purpose, but there are a few worth mentioning:

PublicType—the type of object as it is commonly known and used. Several public types can be built on the same MetadataType.

MetadataType—the type of object in the metadata model that forms the basis of the object.

ObjUri—a string that can be used in DATA step functions to query the SAS Metadata Server for the object. This field is the key for joining the _objs table to the _permsw table.

Permission Columns—each Permission column contains a value indicating the state of the permission as it pertains to the user on this object. The fixed set of values that can appear in this column is as follows:

Granted Explicitly—granted with a white background on the **Authorization** tab.

Denied Explicitly—denied with a white background on the **Authorization** tab.

Granted by ACT—granted with a green background on the **Authorization** tab.

Denied by ACT—denied with a green background on the **Authorization** tab.

Granted Indirectly—granted with a gray background on the **Authorization** tab.

Denied Indirectly—denied with a gray background on the **Authorization** tab.

Granted Explicit Condition—granted with a white background on the **Authorization** tab, but there is a condition associated with the explicit grant of the Read permission. When this appears, there is an entry in the *xxx*_pconds data set containing the condition or condition key.

<*blank*>—generally indicates that the permission does not apply to the object, so no value was extracted.

USING THE SAMPLE SECURITY REPORT—SECRPT.SAS

The data set produced by the macros has detailed information that is in good form for reporting or analysis. It can be queried using SQL or printed for reporting. But, textual reporting can be cumbersome to work through. A better alternative would graphically summarize the grants and denials for all of the users of a single object. SAS is providing a sample application to do just that. The program is SECRPT.SAS and is shipped with the Base SAS samples in 9.2.

Comments within the sample provide documentation for its use. You should make a copy, and then follow the directions for modifying it. Modifications consist primarily of providing the following parameters:

- Metadata connection options
- Path to the directory where the report will be written
- Path of the library where the permission data set or view is located
- Path and filename of the .html file to be created
- Invocation of the %SECRPT macro, including the name of the permission data set and the filter that should be applied to the data set

Figure 1 shows the output generated by the sample program. The sample was executed with the following filter parameter:

```
filter=%str(publictype is not null)
```

This rendered a page of output for each object in the HTML file. You can see from the example that the color-coding that is used on the **Authorization** tab is applied here to indicate the source of the grant or denial. (See Figure 1.) You can also see from the output that the number of permissions that are rendered in the report varies. The sample uses the metadata PublicType dictionary to determine the permissions that are applicable to a specific object based on its PublicType attribute.

When running the sample, you will notice a warning in the log indicating that the **`Data step interface is preproduction for this release`**. That message is generated because the sample takes advantage of a new set of ODS primitive functions in the DATA step. The use of these primitives provides the flexibility needed to render the permission table with object-specific columns, colors, and icons. In our use of these primitives for various permission reports, we have not encountered any issues.

Note: There will likely be thousands of objects in your metadata repository. Care should be taken when designing the extractions and reports to consider resource implications. Extracting all of the objects from a SAS Metadata Server in one request, and then extracting the permissions could be a very resource-intensive process. Furthermore, using the report sample to generate a single HTML file for these objects could yield a file that is too big to be loaded by your browser. It is reasonable to perform large extractions of permission information, and then use the filtering mechanism to produce multiple reports on subsets of the information.

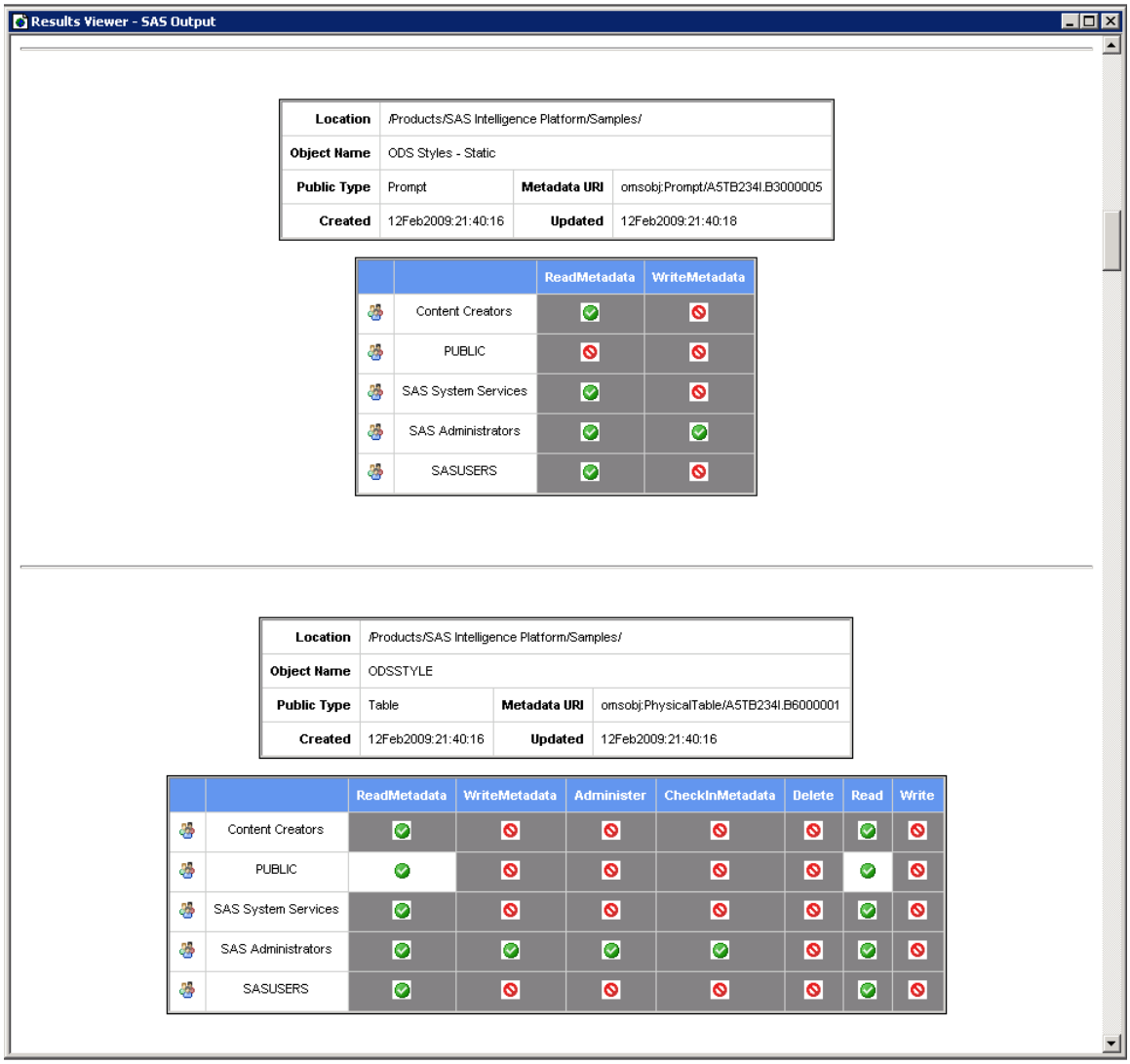


Figure 1. Output Generated by the SECRPT.SAS Sample

Once extracted, any type of SQL WHERE expression can be used in the filter parameter of the sample to subset the data that appears in the reports. You could easily filter to report on just one object, or limit the report to certain users and groups.

REPORTING ONLY ON OBJECTS WITH EXPLICIT OR ACT SETTINGS

Most objects in the SAS Metadata Server inherit their security from a parent object, such as the folder of which they are a member. As a result, most of the objects show the same permission pattern as their siblings in the same folder. Often, it can be useful to have a report that identifies only those objects that have explicit permission settings. These objects are either exceptions when compared to their siblings, or folders that have added restrictions that are passed down to their members and subfolders.

There are several approaches that can be taken to generate such a report. The key is to identify the objects with explicit settings to use as a filter on the report data set. The easiest approach to building this data set is to use a nested SELECT that selects the distinct OBJURIs from the *xxx*_permsl data set. This data set is used because the

permission states are stored in a single column. Regardless of the permissions encountered in a report, this approach will work. Building on the permlib.sampperms extracted in our previous example, submit the following code:

```
proc sql;
  create table permlib.explicitperms as
    select * from permlib.sampperms
        where sampperms.objuri in
           (select distinct objuri from permlib.sampperms_permsl
                where authorization in ("Granted Explicitly",
                                        "Granted Explicit Condition",
                                        "Denied Explicitly",
                                        "Granted by ACT",
                                        "Denied by ACT") );
  quit;
```

The SQL creates a new table—permlib.explicitperms—containing all of the permission state information about all objects on which authorizations have been explicitly administered. Explicit administration includes settings that are provided by an ACT that was applied to the object. Once the data set is created, it can be substituted into the report sample program to generate a report.

IDENTIFYING AUTHORIZATION CHANGES

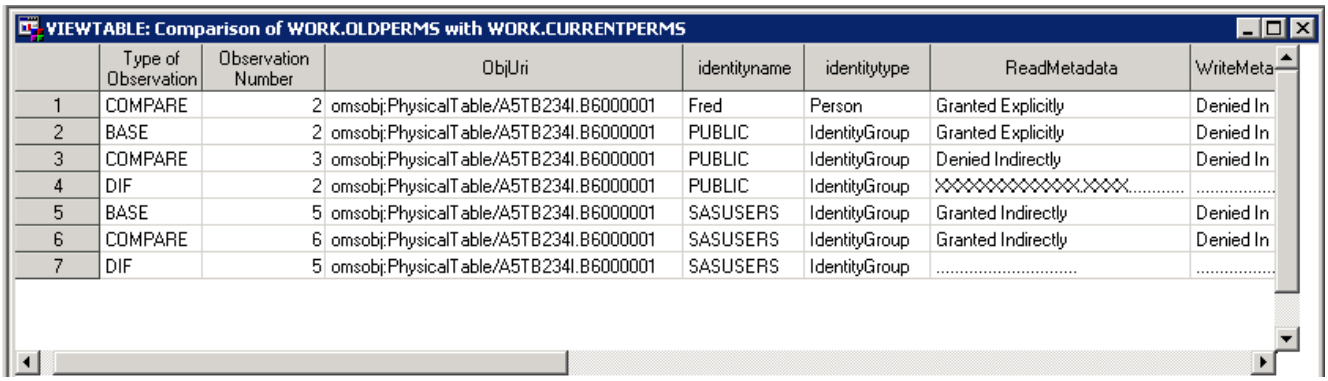
Because the %MDSECGO() macro provides you with a snapshot data set of the security settings at a given point in time, you can use PROC COMPARE to identify changes to security settings that have occurred between two snapshots. In the example code, we use PROC COMPARE with options to create an output data set that contains information only on rows whose permission settings have changed. PROC COMPARE adds information to the data set to identify the source of the differences.

To perform a meaningful comparison, PROC COMPARE requires that the data sets be in sorted order. In the case of our permission comparisons, the keys that are used to identify our records will be ObjIri, Identitytype, and identityname. Rows that have these three columns in common will have their Permission columns compared.

```
proc sort data=oldlib.explicitperms out=work.oldperms;
     by objuri identityname identitytype;
     run;
proc sort data=permlib.explicitperms out=work.currentperms;
     by objuri identityname identitytype;
     run;

proc compare base=work.oldperms
             compare=work.currentperms
             out=permlib.explictchanges outdiff outbase outcomp outnoequal noprint;
     id objuri identityname identitytype;
     vars ReadMetadata WriteMetadata Administer CheckInMetadata
          Delete Read Write Create WriteMemberMetadata;
     run;
```

In the code, you can see that the ID statement is used to specify the columns whose values are used to identify rows that should be compared. The VARS statement informs the procedure which columns in the rows should be compared. In our case, we are interested only in the Permission columns. (It is possible that other permissions are used in your data sets. In that case, adjust the VARS list as appropriate.) Columns that are not identified in the ID statement or the VARS statement are not included in the output data set. This means that information such as the object name, description, metadata type, and so on, needs to be joined back to this data set for reporting purposes. The key variable for joining object information is ObjUri.



VIEWTABLE: Comparison of WORK.OLDPERMS with WORK.CURRENTPERMS

	Type of Observation	Observation Number	ObjUri	identityname	identitytype	ReadMetadata	WriteMeta
1	COMPARE	2	omsobj:PhysicalTable/A5TB234I.B6000001	Fred	Person	Granted Explicitly	Denied In
2	BASE	2	omsobj:PhysicalTable/A5TB234I.B6000001	PUBLIC	IdentityGroup	Granted Explicitly	Denied In
3	COMPARE	3	omsobj:PhysicalTable/A5TB234I.B6000001	PUBLIC	IdentityGroup	Denied Indirectly	Denied In
4	DIF	2	omsobj:PhysicalTable/A5TB234I.B6000001	PUBLIC	IdentityGroup	XXXXXXXXXXXXX.XXXX.....	.....
5	BASE	5	omsobj:PhysicalTable/A5TB234I.B6000001	SASUSERS	IdentityGroup	Granted Indirectly	Denied In
6	COMPARE	6	omsobj:PhysicalTable/A5TB234I.B6000001	SASUSERS	IdentityGroup	Granted Indirectly	Denied In
7	DIF	5	omsobj:PhysicalTable/A5TB234I.B6000001	SASUSERS	IdentityGroup	.....	.....

Figure 2. PROC COMPARE Output Data Set

Figure 2 shows an example of the output produced by the PROC COMPARE code. Two key points can be observed:

- The first column indicates the type of observation. Values include BASE, COMPARE, and DIF. BASE and COMPARE indicate the data set from which the row originated. Rows with a type of DIF are generated by PROC COMPARE. These rows contain markers to indicate which characters between the compared rows are different. Periods indicate matching characters. An X is used to indicate differences. In the fourth row, you see Xs in the ReadMetadata column. If you look at the preceding two rows, you see that the explicit grant of ReadMetadata was removed and the permission is now denied indirectly.
- When a row appears by itself with no corresponding DIF record, it indicates that the user and his permissions were either added or removed in the time between snapshots. If the first column indicates the observation is of type COMPARE, then the identity and its permissions have been added. In the case where the type was set to BASE, the permissions were removed.

The output of PROC COMPARE is useful in that it removes distractions and easily identifies areas of change. It is also in a format that lends itself to reporting. In fact, the security change report that is generated during the 9.1.3 to 9.2 migration is an extension of PROC COMPARE and the sample report discussed in this paper.

REPORTING ON OBJECTS NOT IN FOLDERS

The %MDSECDS() macro generates permission data sets for objects that reside in metadata folders. However, there are some objects that do not reside in folders. For example, users, user groups, roles, ACTs, and various types of server objects are not currently stored in folders. Hence, these objects cannot be reported on using the %MDSECDS() macro. However, the lower-level macros that are called by %MDSECDS() can be used to build permission data sets for these objects. The key element that you must supply is the data set containing the object information that is used to drive the permission extraction.

The SAS code in this example illustrates how to generate the permission data set for server objects defined in the metadata. All of the %MDSEC*xxx*() macros referenced in the source code are included in the autocall macros. Documentation for the macros is included in the macro source. Ask your SAS administrator where the autocall library is located and use a text editor to view the macro source.

As in all of our examples, it is assumed that the metadata connection options have been specified so that you can connect to the server. In the first section of code, we simply assign the library where the data sets are stored and create a template data set. The template is created using %DEFINEOBJTAB_SQL(), which is actually defined in the %MDSECCON() macro. Using this mechanism to create the template table guarantees that the _objs data set we create is in the form expected by the other macros and samples.

```
libname permlib "C:\SAS\EBIserver\Lev1\SASMeta\Data\permdata";

%mdseccon();

/* create empty _objs table */
proc sql;
%defineObjTab_SQL(tabname=work.tmp_objs);
quit;
```

The next section of code uses the template data set in a DATA step to create the _objs data set (permlib.serverperms_objs) that contains information about the various server objects. The metadata DATA step functions are used to retrieve the server objects and their attributes, which are then output into the data set. The key to the DATA step is the first parameter specified in the METADATA_GETNOBJ() function. This parameter is used to specify the type of objects and filter criteria. In the case of application servers, logical servers, and server components, all of these objects include sub-classes of the ServerComponent metadata type and each has a PublicType attribute that contains the word 'Server.' When the objects that meet this criteria are returned, we loop through each one and retrieve the attributes needed for the _objs table. The output statement ensures that the information about each object is added as a row in the data set.

Note: The `if 0 then set work.tmp_objs;` statement prevents the SET statement from executing. However, the variables in the template data set are still defined.

```
data permlib.serverperms_objs(drop=n nobj rc);
    if 0 then set work.tmp_objs;
    nobj=1; n=1;

    nobj=metadata_getnobj(
             "omsobj:ServerComponent?@PublicType contains 'Server'",n,ObjUri);

    do while(nobj >= 0);
       rc=metadata_resolve(ObjUri, MetadataType, ObjId);
       rc=metadata_getattr(ObjUri,"Name", ObjName);
       rc=metadata_getattr(ObjUri,"PublicType",PublicType);
       rc=metadata_getattr(ObjUri,"Desc",Desc);
       rc=metadata_getattr(ObjUri,"MetadataCreated",MetadataCreated);
       rc=metadata_getattr(ObjUri,"MetadataUpdated",MetadataUpdated);

       output;

       n=n+1;
       nobj=metadata_getnobj(
                "omsobj:ServerComponent?@PublicType contains 'Server'",n,ObjUri);
    end;
    stop;
run;
```

Now that the _objs data set has been created, a few simple macros can be called to drive the rest of the process before using a DATA step to build the final data set used for reporting. The macros perform the following functions:

%MDESCGP()

For each object in the _objs data set, this macro retrieves all of the permission settings. The resulting _permsl data set contains the permission state for a single permission for a specific user or group in each row.

%MDSECTR()

This macro is used to transform the _permsl data set into a _permsw data set. Each permission becomes a column in the output table.

%MDSECVW()

This macro generates a view that joins the information in the _objs table with the permission information in the _permsw table.

```
/* Call to the mdsecgp macro  */
%mdsecgp( objdata=permlib.serverperms_objs, permdata=permlib.serverperms_permsl,
          conddata=permlib.serverperms_pconds );

/* Call to the mdsectr macro  */
%mdsectr( permdata=permlib.serverperms_permsl, trandata=permlib.serverperms_permsw );

/* Call to the mdsecvw macro  */
%mdsecvw( objdata=permlib.serverperms_objs, trandata=permlib.serverperms_permsw,
          targdata=permlib.serverperms_join  );
```

```
data permlib.serverperms;
   set permlib.serverperms_join;
   run;
```

The resulting data set can be used with the sample security report. (See Figure 3.) The location for each object is empty. Note the additional permissions applied to the SASTS – Logical Table Server. This server enforces SQL permissions for the data it contains. Because of the PublicType information about the object, the report was able to automatically display this information without additional modification.

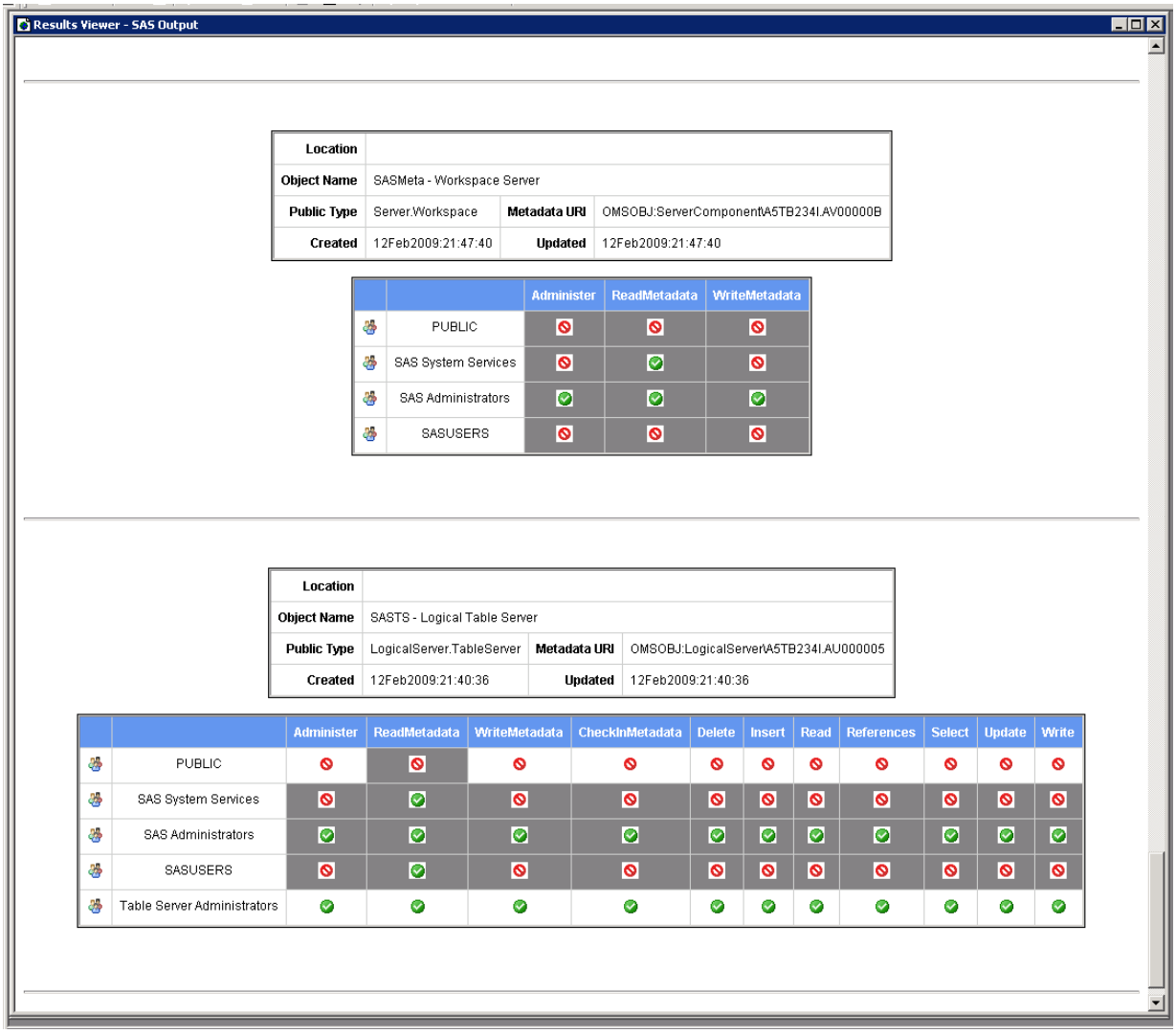


Figure 3. Example Security Report for Servers

CONCLUSION

The security auditing and reporting capabilities of the 9.2 SAS Metadata Server are a tremendous asset for SAS administrators. The default settings of the logging facility provide audit records for any action that affects metadata authorization decisions. Changes that affect users, groups, logins, access control changes, and access control templates are automatically logged and easy to locate. Configuration is flexible and dynamic. Viewing the active server log and changing logging levels from SAS Management Console provides administrators with the ability to monitor server activity unobtrusively while it occurs.

The reporting macros are simple to use and can harvest a huge amount of accurate information about the authorization settings in the server. Having this information in data sets enables you to easily report on subsets of the extracted data or to analyze content and identify changes that have occurred over time.

The information discussed in this paper should give you insight into how these features work and how you can use them to know your system. To that knowledge, you can apply your intelligence to better monitor and manage your system with confidence.

ACKNOWLEDGMENTS

Thanks to Suresh Ramakrishnan and Brian Bowman for providing research and background information about the auditing capabilities in the SAS Metadata Server.

RECOMMENDED READING

Chapter 10, “Security Report Macros,” in *SAS 9.2 Intelligence Platform: Security Administration Guide*

Chapter 7, “Monitoring the Activity of SAS Servers,” and Chapter 8, “Administering Logging for SAS Servers,” in *SAS 9.2 Intelligence Platform: System Administration Guide*

SAS 9.2 Logging Configuration and Programming Reference

Part 5, “DATA Step Functions,” in *SAS 9.2 Language Interfaces to Metadata*

CONTACT INFORMATION

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

Forrest Boozer
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
Forrest.Boozer@sas.com

Diane Hatcher
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
Diane.Hatcher@sas.com