Tuning RACF Generics

Lennie Dymoke-Bradshaw
Infrastructure Consultancy & Services
ITS Services Business
Global Services
IBM UK Ltd.

Version 1.3
27 June 2006

IBM United Kingdom Limited
Greenford Green Business Park
Green Park Way
GREENFORD
Middlesex UB6 0AD

Document:Tuning Racf Generics 005.lwp

Acknowledgements

Thanks are due to the following IBMers for their knowing or unknowing assistance in the development of this document.

Ken Warren	-	For suggestions and improvements
Walt Farrell	-	For suggestions and improvements

Contents

Figures

1. Introduction

RACF has provided protection of resources using generic profiles since RACF 1.5 was introduced in the mid-1980s. It has proved so popular that few installations now use the predecessor mechanism of discrete profiles.

However, few users have actually considered the effect of the use of generic profiles on the performance of their installation. Neither have there been many studies on how to improve the efficiencies of these generic profiles.

This document attempts to remedy this.

The document draws on the author's experience in working with RACF in an MVS environment since the late 1970s.

The intended audience for this document are MVS systems programmers and MVS security administrators. While some attempt is made to explain some of the control blocks in use the reader is expected to know many terms related to both MVS and RACF.

References to RACF apply also to the RACF component of the OS/390 Security Server and the RACF component of the OS/390 Secureway Security Server.

2. Generic Profiles - What are they?

This section gives some background information on the structure of RACF use within an address space. Inevitably this involves some discussion of control blocks. While a glossary of terms is available at the end of this document, the reader is referred to other texts on MVS for more details.

Most of the discussion here centres on a fairly normal address space; i.e. one used by a TSO user.

2.1. ASCB to ASXB to ACEE

Each TSO user, started task, batch job resides in an address space. On MVS/ESA systems this can address up to 2Gb of storage (31 bits can address 2^{31}bytes). Of this storage, some is “common”, that is it can be addressed from all address spaces. Typically this applies to the PSA (Prefixed Storage Area) which is the first 4Kb of each address space, to the nucleus (which straddles the 16Mb line) and to some other storage areas around the Nucleus.

Contained within common storage are the Address Space Control Blocks for each address space. There is one ASCB for each address space, and they are all stored in common storage. Thus they can all be seen from all address spaces.

From each ASCB is a pointer to the ASXB (Address Space eXtension Block). Unlike the ASCB, the ASXB is held in “local” storage. This means that it cannot normally be addressed from other address spaces.

From the ASXB is a pointer to another control block called the Access Control Environment Element. (ACEE). The ACEE is a key part of the RACF architecture. This control block represents the identity of a user, and when it is pointed to from the ASXB it controls the identity of the address space. It is used to control what resources can be accessed by the user operating in the address space.

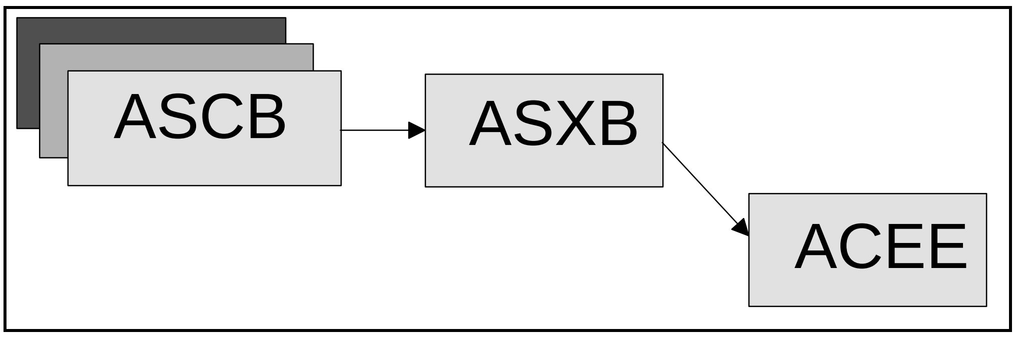

Figure 1 ASCB to ACEE

The ACEE is key to the security of the address space. It is normally created with a RACROUTE macro using the REQUEST=VERIFY parameter. This is issued on behalf of the user when he or she signs on to TSO, when a batch job starts for him, or by Started task Control for a new started task.

Contained within the ACEE (and some areas pointed to by it) are the users userid, the name of the users default group, the users name and any special attributes he or she has, such as OPERATIONS, SPECIAL etc. Also in here are pointers to further control blocks used in other security mechanisms.

I will proceed now to understand what happen when a user uses a resource protected by a discrete profile. But first a discussion of fastpath mechanisms.

2.2. Fastpath mechanisms

By “Fastpath mechanisms” I mean any process which would cause RACF checking to be done prior to reading discrete or generic profiles. In the sections below, on discrete profile processing, and generic profile processing I am assuming that no special mechanism has been used to take an “early” decision.

There are several methods which may be used to take an early decision. I list these in no particular order.

2.2. 1. Priviledged flag

The Priviledged flag can be set in the Started Task table ICHRIN03 for a started task, or may be set using the STARTED class, also for a started task. If this bit is set then all RACROUTE REQUEST=AUTH macros will give return code 0, thus indicating that access should be granted.

The Priviledged flag is usually used for system started tasks only. It is set in the started task control table ICHRIN03 or via the STARTED RACF class.

2.2. 2. Trusted flag

This flag is similar to the Priviledged Flag above, but differs in that auditting flags will be honoured.

The Trusted flag is usually used for system started tasks only. It is set in the started task control table ICHRIN03 or via the STARTED RACF class.

2.2. 3. System Exit

The main system exit which may grant access prior to checking profiles is the RACROUTE REQUEST=AUTH pre-processing exit ICHRCX01. If this exit gives a return code of 8 or 12, then access will be granted without checking profiles.

2.2. 4. GLOBAL profiles

If a GLOBAL profile exists in storage which matches the resource in question then this can grant access to the resource without needing to access the profiles.

2.2. 5. NOPASS

This option may be set within the SCHEDxx member of SYS1.PARMLIB. It is an instruction to data management to avoid issuing the RACROUTE REQUEST=AUTH for dataset OPENs. While this differs from the other fastpath mechanisms above insofar as it is not a RACF mechanism, it does effectively prevent RACF from using discrete or generic profiles.

2.3. Discrete Profile mechanism

We will take a example of a user opening a dataset which is protected by a discrete profile. (This discussion assumes the dataset is a non-VSAM dataset[1]). DFSMS will use a discrete profile to protect a dataset if a bit is set on in the DSCB for the dataset (The DSCB is a control block that resides on a DASD volume. The existence of a DSCB virtually defines the existence of the dataset.)

This bit (the DS1IND40 bit) is set on when a dataset is defined by a DD statement specifying PROTECT=YES, when created by a user who has the ADSP attribute, or when a user issues the ADDSD command for the dataset.

When the user attempts to open the dataset, DFSMS issues a RACROUTE REQUEST=AUTH macro which attempts to find the RACF profile which protects the dataset. DFSMS checks the DSIND40 bit, and finding that it is set on, uses a parameter on the RACROUTE macro called RACFIND=YES. (This parameter effectively tells RACF that this dataset is protected by a discrete profile rather than a generic profile).

Before searching the RACF database RACF makes some preliminary checks. These include determining if the ‘Privileged' or ‘Trusted’ bits are set on. These bits are normally not set for TSO address space so the next check is to see if a GLOBAL profile covers this dataset. Let us assume that in this case there is no such GLOBAL profile.

In order to find the profile which matches the dataset name the RACF database is searched, the profile is brought into storage, and then a matching process takes place to determine whether this user (as defined by the ACEE) has the required level of access as shown in the profile.

Once the decision has been made (and we are not really concerned what that decision was) the profile is discarded; that is the storage in which it resides is discarded, or freed.

The process of searching for the profile makes use of any in-storage control blocks which have been set up by the systems programmer using the values in the RACF Dataset Name Table known as ICHRDSNT.

You will see that once the process is complete, the ACEE and other storage are much the same as they were before the check took place.

2.4. Generic Profile mechanism

Now let us suppose that the dataset mentioned above was not protected by a discrete profile.

In this case, it was not created by a user with ADSP, nor was PROTECT=YES specified on the DD card which created it, nor was an ADDSD issued for it once it was created.

Instead, it was just created by a RACF user, and later someone tried to open it.

In this case DFSMS will examine the DSIND40 bit and find it set off. Hence DFSMS issues a RACROUTE REQUEST=AUTH macro as before, but this time RACFIND=NO is specified.

RACFIND=NO is a signal to RACF that this resource is protected by a generic profile. At this point, not only does RACF not know which generic profile protects this resource, RACF does not even know if there is a profile that protects it at all. What is needed is to go through some kind of matching process to find the correct profile.

[1] Similar processing takes place for VSAM datasets, but the bit corresponding to the DS1IND40 bit is held in the catalog.

This is what actually happens.

Let us assume that the dataset name of this dataset is:-

```
    HLQ01.TEST.DATA
```

RACF will now read the database and bring into storage the names of all generic profiles which have the same high-level qualifier (or HLQ) as this dataset. So in this case RACF reads the database and find the following profiles:-

```
    HLQ01.A.** (G)
    HLQ01.B.CLIST (G)
    HLQ01.B.TEST%% (G)
    HLQ01.TEST.** (G)
    HLQ01.** (G)
```

The (G) after each profile denotes that this is a generic profile. This is important in the case of the 2nd profile above, as it could also be the valid name of a discrete profile.)

RACF then matches our dataset name against the list to determine the best match. In this case it is the fourth profile

```
    HLQ01.TEST.** (G)
```

and this profile is used to make the security decision. However, let us look more carefully at what actually happens in storage.

2.4. 1. Generic profile control blocks

When RACF receives a RACROUTE REQUEST=AUTH with RACFIND=NO specified (and this is a check for a dataset profile) first some preliminary checks are made; the same as for a discrete profile (privileged bit, trusted bit, GLOBAL profiles, etc.).

Let us assume that none of these cause an early decision to be taken. RACF knows that Generic profiles must be used to determine the access decision.

A field in the ACEE is now examined. This is the ACEEGATA field. This can point to sets of in-storage generic profiles. These profiles are organised in sets according to the high-level qualifier of the dataset. So RACF must now look for a set with a high-level qualifier of “HLQ01”.

Let us assume that no such list was found. Consequently RACF now must construct such a list. RACF does this by reading the names of **all** the generic profiles with a HLQ of HLQ01 and moving these into a structure in storage. This structure is known as a Generic Anchor Table Entry or GATE. Each GATE can point to another GATE, so that several can be storage at the same time. However, crucially, there can only be a maximum of four GATEs for each ACEE at a time[2].

Once the GATE is built in storage, it has pointers to the names of all the generic profiles for the high level qualifier. For each name there is a pointer which can address an actual in-storage profile. However these are not all brought into storage at once.

Once the GATE is built, it is searched to determine the best match. Once the best match has been determined, the pointer to the profile is accessed. If the profile is not present (as it will not be the first time through) the RACF manager is invoked to read the profile into storage and the pointer is updated to point t that profile.

RACF can now make a decision on whether the access is allowed, but I have said before, this is of little interest to us.

[2] While there have been requests made to change this number over many years, it has remained constant since generic profiles were introdcued.

2.4. 2. The 4 Gates

Figure 2 above shows the GATEs in storage, with each one pointing to multiple GPRFs. (GPRF is the name of the control block containing the in-storage profile.)

Given that only 4 of these GATEs are allowed in storage at any one time, one might ask what happens when the requirement arises for a fifth GATE.

At this point the GATE which was last used the longest ago is discarded (along with any generic profiles pointed to from it), and the new GATE is built and chained to the others.

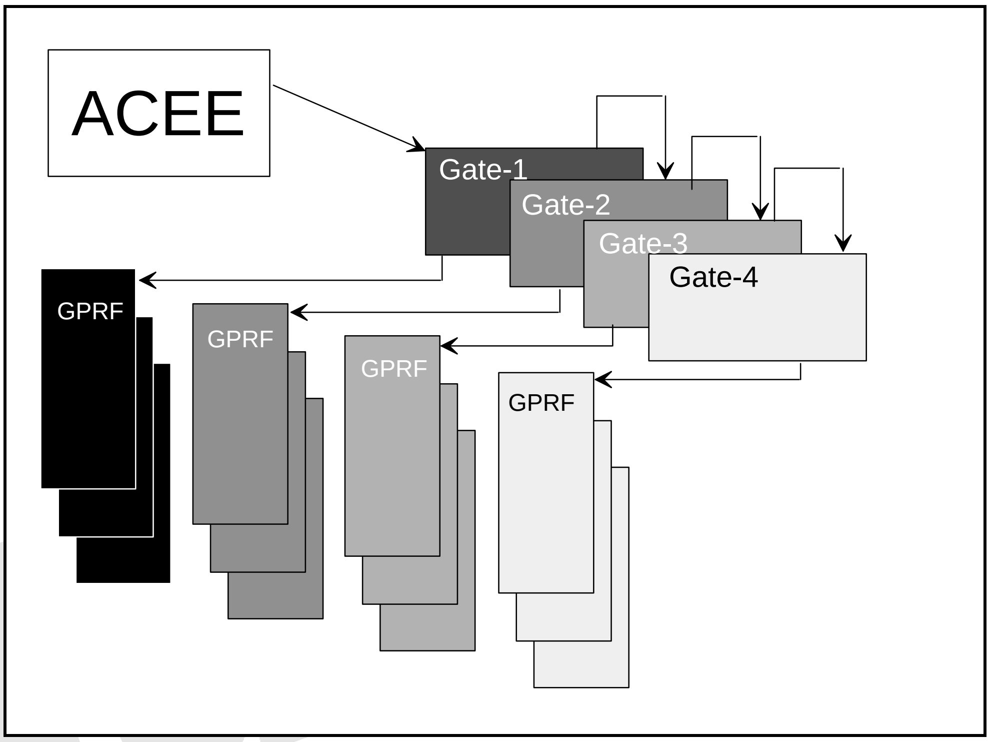

Figure 2 GATEs in storage

An important point here is that all new GATEs are built at the front of the chain of 4, and each time a GATE is used (even if it was already in existence) it is moved to the front of the chain if not already there. Thus the GATE which was last used the longest ago will be the one at the end of the chain.

If access is attempted to resources which are not datasets (and which have not been brought into storage using a RACLIST function - see later) then a similar structure is built in storage, but this time the entire class of profiles is brought into storage. Thus if access is attempted to a profile in the SDSF class, then all profiles names in that class are brought into storage and a GATE and GPRF structure built to represent them. Note that this GATE and GPRF structure is included in the set of four.

You will see that it is possible for the system to spend much time manipulating the sets of profile names addressed by the GATEs.

The RACF command:-

```
SETROPTS GENERIC(DATASET) REFRESH
```

can be used to cause all dataset generic profiles to be flushed at the next attempt to use generic profiles.

If the security administrator decides to refresh generic profiles in a given class using this SETROPTS command then this might result in a lot of extra activity for each address space as it rereads the profiles, and rebuilds the in-storage structure at the next access.

2.4. 3. Storage Considerations

The GPRF control block is built in subpool 225 ELSQA[3]. It resides in the local address space.

From experience the amount of storage taken by each generic profile name that is in storage is about 36 bytes. This is merely a rule-of-thumb and is obviously affected by many things, including the length of the names, the type of generic structures used and how many pages are required[4].

In a recent examination of a customer I found a class in use similar to the JESSPOOL class. This class had more than 10,000 profiles.

Each time any profile in this class was accessed all 10,000 profiles had to be brought into storage. This resulted in around 352K of storage being used in each address space for these profiles. There were around 400 TSO users on this system, and little paging. So the storage occupied by these profiles was around 138Mb of real storage.

2.4. 4. Thrashing

Let us call each set of profile names stored under a GATE a Generic Profile Set.

If an address space makes use of more than 4 Generic Profile Sets on a regular basis, then we move to a situation where the GATEs are thrashing; that is they spend a lot of time being discarded and being rebuilt in storage. The higher the number regularly accessed the worse the situation.

Also if the sets are very large (as in the example above) you will see that a great deal of processing has to take place merely to continue accessing resources.

The management of storage for these GATEs and GPRFs does not take into account how much work is done rebuilding the structures. If a GATE is the oldest one in storage and its slot is needed, it is discarded, whether it has 1 profile or 10,000.

The later discussion of CLIST concatenations in section 2.2.2.1 shows how bad order selection of datasets, and poorly chosen naming standards can have a very bad effect on GATE performance.

[3] It could conceivably be built in LSQA rather than ELSQA. This might happen if system exits were in use which worked with an AMODE of 24. This is highly undesirable, and would usually be avoided.
[4] Storage is managed at the byte level by the virtual storage manager, but the operating system moves pages around, which are 4K in size.

3. Tuning Generic Profiles

This section looks at the mechanisms and ideas we can use to avoid thrashing the GATEs. Some of these ideas can be used in most installations, other parts may require more detailed planning.

The thrust of most of this is to try and move processing away from generic profiles. If a user only rarely uses more than 4 GATEs, then his processing will be far more efficient. If 5 are used, then processing can be very inefficient, depending on access patterns.

Removing one generic profile set can have a very large knock on effect on all his generic profile processing.

3.1. SETROPTS options

The most fundamental approach to improving performance is to avoid doing work. The first approach therefore is to see what can be done to avoid reading the profiles into the GATEs

3.1. 1. GLOBAL profiles

Global profiles are used to give access to resources. They are never used to deny access. If you have some resource, dataset or otherwise which you can simply identify and which is used a great deal, then create a GLOBAL profile for it.

Let us suppose for example an installation uses datasets having a HLQ of SYS2 and there are 20 generic profiles protecting that set. Also all users need to access the dataset:-

```
SYS2.CLIST
```

on a regular basis for READ access. However they do not need to access other datasets which have a high level qualifier of SYS2.

The security administrator can create a GLOBAL profile for the dataset class with the following command:-

```
REDEFINE GLOBAL DATASET UACC(NONE)
```

and he can add the profile for SYS2.CLIST to it as follows:-

```
RALTER GLOBAL DATASET ADDMEM('SYS2.CLIST'/READ)
```

The profiles are then loaded into storage with the command:-

```
SETROPTS GLOBAL(DATASET)
```

Once this has been done all READ access requests for SYS2.CLIST will be granted without generic profiles being used. However, if a user does need to access other profiles within the SYS2 set, then the normal processing will take place. If the user needs to update

SYS2.CLIST then the GLOBAL profile does not handle update access so the normal generic profile processing mechanism will be used.

To summarise, there is no loss of integrity[5] with GLOBAL profiles, but there can be an enormous gain in efficiency for all TSO users using any dataset that the GLOBAL profile covers.

3.1. 2. RACLISTed classes

This technique can be used for general resource classes, and cannot be used for datasets. However, as we have seen, if this frees up a GATE entry then there can be benefits for dataset processing too, as there will be less thrashing of the GATEs.

A general resource class can have all of its profile names loaded into a data space, so that all users can access the same set of in-storage profiles. This is known as RACLISTing the class.

When the class is RACLISTed a dataspace[6] is created which contains the same structure of profile names and profiles. However it is not necessary for each address space to have a copy of the profiles, and so there is no requirment to use up one of the GATE entries.

The profiles in the dataspaces can be refreshed by the security administrator using the SETROPTS command.

Let us suppose that the system above uses SDSF for accessing the JES2 spool, and wants to use SDSF classes to protect various features within SDSF.

There are usually quite a lot of these profiles, and many sites also use the JESSPOOL class as well. If these are just defined and used and no other action taken, you will see that this will take up 2 GATE entries, and will very likely cause dataset GATEs to be thrown out and then subsequently rebuilt when needed. Of course when they are rebuilt, some of the GATEs needed for SDSF will be thrown out again, and so on. This will result in a lot of unnecessary processing.

In this case we will move all of the profiles in these three classes into dataspaces. This can be done with the following commands:-

```
SETROPTS RACLIST(SDSF)
SETROPTS RACLIST(JESSPOOL)
```

Once this has been done, any change to a profile in these classes will cause a message to be issued stating that the change will not be effective until a REFRESH is issued. The refresh can be done as follows:-

```
SETROPTS RACLIST(SDSF) REFRESH
```

However, this last command will only cause one set of I/O activity. This should be contrasted with a command such as:-

```
SETROPT GENERIC(DATASET) REFRESH
```

which will potentially cause all address space to cast out in-storage generic profile lists, and rebuild them.

Remember that you could also make use of GLOBAL profiles for these classes too. If so, the global checking would take place prior to checking the RACLISTed profiles.

[5] Unless you are using a security classification for this resource. If so, do not use a GLOBAL profile for the resource.
[6] The dataspace will be owned by the *MASTER* address space, but will be accessible from all address spaces. They can be displayed, but not identified using the MVS command “D A,*MASTER*. The dataspaces for RACF can be seen displayed. They are the ones with names starting IRR.

3.2. Naming standards

3.2. 1. Catalogues

You will see that once you have moved most of your general resource class profiles into RACLISTed status, what you are left with are datasets.

You should look at these carefully. You may find, for example that you are frequently opening and searching catalogues. If your catalogues all start with a common HLQ (many installations simply use an HLQ of CATALOG) then this is a good candidate for a GLOBAL profile. Some sites however do not allow users to search the master catalog. If this is your wish you can set up GLOBAL profiles to allow READ access to all catalogs except the master as follows.

I will assume there are already some GLOBAL profiles in existence.

Issue,

```
RALTER GLOBAL DATASET ADDMEM(‘CATALOG.MASTER’/NONE)
RALTER GLOBAL DATASET ADDMEM(‘CATALOG.**’/READ)
```

3.2. 2. TSO/ISPF system datasets

Many installations use multiple HLQs for their live TSO/ISPF environment. Let us take a look at such a system and see where the problems lie from a RACF generic perspective.

I recently came across a CLIST which was invoked at each TSO logon. An extract is shown in figure 3.

```
/*********************************************************************/
/* ALLOCATE SYSPROC DATASETS                                         */
/*********************************************************************/
FREE FI(SYSPROC)
ALLOC FI(SYSPROC) SHR DA( +
                         'SYS1.SBLSCLI0'          /* IPCS        */  +
                         'BDT.SBDTCLI0'           /* BDT         */  +
                         'BOOKMAN.SEOYCLIB'       /* BOOKMAN     */  +
                         'CBC.SCBCUTL'            /* C++         */  +
                         'DFSORT.SICECLIB'        /* DFSORT      */  +
                         'SYS1.DGTCLIB'           /* DFSMS       */  +
                         'FFST.SEPWCENU'          /* FFST        */  +
                         'SYS1.SCBDCLST'          /* HCD         */  +
                         'ISP.SISPCLIB'           /* ISPF        */  +
                         'RMF.SERBCLS'            /* RMF         */  +
                         'SYS1.HRFCLST'           /* RACF        */  +
                         'GIM.SGIMCLS0'           /* SMP/E       */  +
                         'ICQ.ICQCCLIB')          /* TSO/E       */
```

Figure 3   Extract from Logon Clist

Let us consider what would happen when the user attempts to invoke a CLIST via different methods.

3.2. 1. Implicit execution

By implicit execution I am talking of the user simply issuing a command by typing its name at the READY prompt, by typing its name on option 6 of ISPF, or of prefixing it with TSO and typing this on the command line of a panel.

In such an instance the system would first search for the name as a TSO command. This would involve a search for the command as a load module. Let us assume that this is unsuccessful.[7]

At this point TSO needs to search for the command as a CLIST or REXX exec. The first step on this process is to open the SYSPROC file[8].

It is now that the problem becomes apparent for there are 10 different high-level qualifiers in use for the concatenated set of datasets. Thus, opening the SYSPROC file will involve throwing out all existing generic profiles, and replacing them with others.

Each time that SYSPROC is opened this will involve the same processing.

This concatenation of datasets with many different high-level qualifiers would be exacerbated by having a lot of generics on some of the high-level qualifiers.

While there are actions that can be taken to help this situation it is best avoided by using less high level qualifiers.

But let us consider in more detail what hapens during an OPEN of the SYSPROC Ddname shown above. During the OPEN of a concatenated dataset, an open is issued for each dataset in turn; and this open issues a RACROUTE REQUEST=AUTH which will drive the GATE processing.

Table 4 illustrates how profiles can be loaded in and subsequently discarded after only using one name once. Further, the names left in the 4 GATEs at the end of processing may not be the ones that will be of the most use in future. For example, if the clist execution is followed by another of a different name, only one of the sets of names will remain in storage (the SYS1 set).

Dataset to Open	Action	Gate 1	Gate 2	Gate 3	Gate 4	Consequent Action
SYS1.SBLSCLI0	Load SYS1 names	SYS1	#1	#2	#3	Delete #4 names
BDT.SBDTCLI0	Load BDT names	BDT	SYS1	#1	#2	Delete #3 names
BOOKMAN.SEOYCLIB	Load BOOKMAN names	BOOKMAN	BDT	SYS1	#1	Delete #2 names
CBC.SCBCUTL	Load CBC names	CBC	BOOKMAN	BDT	SYS1	Delete #1 names
DFSORT.SICELIB	Load DFSORT names	DFSORT	CBC	BOOKMAN	BDT	Delete SYS1 names
SYS1.DGTCLIB	Load SYS1 names	SYS1	DFSORT	CBC	BOOKMAN	Delete BDT names
FFST.SEPWCENU	Load FFST names	FFST	SYS1	DFSORT	CBC	Delete BOOKMAN names
SYS1.SCBDCLST	Use SYS1 names	SYS1	FFST	DFSORT	CBC	none
ISP.SISPCLIB	Load ISP names	ISP	SYS1	FFST	DFSORT	Delete CBC names
RMF.SERBCLS	Load RMF names	RMF	ISP	SYS1	FFST	Delete DFSORT names

7 It would be preferable if the command had been prefixed with a % mark to avoid this search.
8 Assuming SYSEXEC is not allocated.

Dataset to Open	Action	Gate 1	Gate 2	Gate 3	Gate 4	Consequent Action
SYS1.HRFCLST	Use SYS1 names	SYS1	RMF	ISP	FFST	none
GIM.SGIMCLS0	Load GIM names	GIM	SYS1	RMF	ISP	Delete FFST names
ICQ.ICQCCLIB	Load ICQ names	ISQ	GIM	SYS1	RMF	Delete ISP names

Figure 4 Actions during a SYSPROC open.

This situation is made far worse than it need be by DFSORT.SICELIB having been placed before SYS1.DGTCLIB, as the SYS1 names are deleted and then subsequently reloaded. (Items shaded in table 4).

Table 5 illustrates how a simple change in the order of these two libraries will avoid the profiles being discarded and then subsequently reloaded, at the next access check.

While this chart shows the actions taking place for each access check, they do not show the amount of work being done.

For example, if there are many generic profiles for the SYS1 high-level qualifier, say 100, then the work done to read them into storage (and subsequently to discard them) is greater than if there had been a smaller number. This shows what I have mentioned before under the section on Thrashing. While the LRU (Least Recently Used) algorithm works to keep in storage the sets of profiles names which have been most recently used, it takes no account of the number of names involved in each set.

Dataset to Open	Action	Gate 1	Gate 2	Gate 3	Gate 4	Consequent Action
SYS1.SBLSCLI0	Load SYS1 names	SYS1	#1	#2	#3	Delete #4 names
BDT.SBDTCLI0	Load BDT names	BDT	SYS1	#1	#2	Delete #3 names
BOOKMAN.SEOYCLIB	Load BOOKMAN names	BOOKMAN	BDT	SYS1	#1	Delete #2 names
CBC.SCBCUTL	Load CBC names	CBC	BOOKMAN	BDT	SYS1	Delete #1 names
SYS1.DGTCLIB	Use SYS1 names	SYS1	CBC	BOOKMAN	BDT	none
DFSORT.SICELIB	Load DFSORT names	DFSORT	SYS1	CBC	BOOKMAN	Delete BDT names
FFST.SEPWCENU	Load FFST names	FFST	DFSORT	SYS1	CBC	Delete BOOKMAN names
SYS1.SCBDCLST	Use SYS1 names	SYS1	FFST	DFSORT	CBC	none
ISP.SISPCLIB	Load ISP names	ISP	SYS1	FFST	DFSORT	Delete CBC names
RMF.SERBCLS	Load RMF names	RMF	ISP	SYS1	FFST	Delete DFSORT names
SYS1.HRFCLST	Use SYS1 names	SYS1	RMF	ISP	FFST	none
GIM.SGIMCLS0	Load GIM names	GIM	SYS1	RMF	ISP	Delete FFST names
ICQ.ICQCCLIB	Load ICQ names	ISQ	GIM	SYS1	RMF	Delete ISP names

Figure 5 Actions during a SYSPROC open (slightly optimised)

In order to avoid these situations datasets with like high-level qualifiers can be placed next to one another, or with no more than 3 other dataset high-level qualifiers between them.

However one should also note that there are other peformance considerations with such CLIST concatenations. The placement of more frequently used datasets near the top of the concatenation is generally good in terms of avoiding directory searches.

This situation is helped by the use of VLF management of CLIST library directories, but worsened by the use of a SYSEXEC DD statement which is usually searched ahead of SYSPROC.

Other mechanisms which can be used to assist are the ALTLIB command and explicit invocation of CLISTs.

3.2. 2. Explicit invocation

When a CLIST is implicitly invoked a command similar to that below will be used:-

```
    EXEC ‘SYS1.CLIST(FRED)’
```

This will ensure that only the dataset in question is opened, so that only its high-level qualifier is added to the GATE chain.

However, while we have optimised this part of the invocation process, an allocation must still take place for the dataset ‘SYS1.CLIST’. Furthermore it is necessary to know exactly where the CLIST routine resides, and to know if it is moved. It is for this reason that many installations make more extensive use of implicit execution.

3.2. 3. ALTLIB command

If you are invoking a dialog which needs to access many CLISTs within one library, then you may use the ALTLIB TSO command to direct just one library to be searched as follows:-

```
    ALTLIB ACTIVATE APPLICATION(CLIST) DA(‘SYS2.CLIST’)
```

ALTLIB can manipulate application level and user level libraries.

You may also wish to examine the EXECUTIL command for more details. Each of these commands can be found in TSO/E Command Reference.

3.2. 4. An alternative approach

My preferred approach to this is to use a ‘merge’ mechanism to create one large CLIST library from all those needed to supply the PDF services needed. This obviously needs to be a carefully controlled process, and needs to be a ‘defined’ rebuild, by which I mean that all the CLISTs should be maintained in their respective libraries (under the names shown in the above logon CLIST extract for example) but that they are copied into one large CLIST library for execution purposes.[9]

This simplifies the VLF situation, both in terms of the number of libraries managed by VLF, and in terms of avoiding the need to add new libraries to the VLF definitions in the COFVLFxx member.

This same process can then be used for other sets of datasets required for TSO sessions. This includes ISPPLIB, ISPMLIB, ISPSLIB and others.

[9] This also speeds the member search process, as only one directory needs to be searched, and so only one channel program need be built to search it. The worst case situation for a large concatenated set is always when the command name has been typed incorrectly, and so all libraries must be searched. If you have 10 libraries then 10 channel programs must be used. If you have only one library, then only 1 channel program is needed.

3.3. Real-time changes not implemented

A by-product of the way that generic profile processing works in individual address spaces is that sometimes changes that are implemented are not reflected into those address spaces until it is necessary to bring in the profiles again.

If profiles are changed in storage very frequently, (as will happen if many HLQs are in use) then real-time changes are picked up quickly.

There are several approaches to managing this situation.

For a change affecting an individual user, (particularly when granting access) the user is usually asked to logoff and logon again. This causes all the profiles to be discarded and rebuilt.

For a change affecting many users, or for one which must be implemented quickly (more likely when the administrator wants to deny access to a resource), the administrator might issue:-

```
    SETROPTS GENERIC(DATASET) REFRESH
```

This command is effective, but potentially very expensive. It will cause all GATEs in all address spaces to be rebuilt at the next GATE access. Frequent use of the above command is very inadvisable. See information apar II11017 for more details.

Another approach to this is the use of the command LISTDSD.

If the command LISTDSD DA('dataset.name') GENERIC is used then the set of names for the corresponding high-level qualifier will be refreshed. Unfortunately this cannot be used for non-dataset profiles.

4. Recommendations

The following set of recommendations can be used to improve the performance of generic profiles, but should not be taken in isolation. Most of these mechanisms are discussed in the foregoing chapters.

1 Examine all use of general resource classes, especially those used in TSO address spaces. Classes used for SDSF (or (E)JES if you use JES3) are very good candidates for being RACLISTed using the SETROPTS command.

2 Examine your use of GLOBAL profiles. Consider placing your commonly used ISPF and TSO datasets into the DATASET GLOBAL profile with READ access. Consider also placing your CATALOGs into the DATSET GLOBAL profile.

3 Examine your TSO logon JCL and any CLIST used to allocate libraries for your TSO sessions. Consider the order of the libraries on the SYSPROC and SYSEXEC DD names with regard to the discussion in chapter 3. If there is no known reason for the order you have, consider grouping high-level qualifiers. Try and place the most heavily used libraries at the top. Remember that the most heavily used libraries will not necessarily be the largest.

4 Examine your dataset naming standards, particularly those used by your TSO and batch users. If you make use of a multitude of high-level qualifiers which could be reduced then do so.

5 Consider a merged approach to your common TSO/ISPF datasets.

By copying all the CLIST libraries shown in Figure 3 into one library, you could avoid nearly all of the RACF processing shown in Figure 4 and Figure 5.

This same approach can be taken with your ISPF datasets, i.e. the datasets on the following DD names: ISPPLIB, ISPLLIB, ISPMLIB, ISPTLIB, ISPSLIB, SYSHELP.

Glossary of terms

ASCB	Address Space Control Block. The primary control block defining the existence of an address space. Held in common storage.
ACEE	Accessor Control Environment Element. A control block which normally resides within each address space and defines the user of the address space.
ACEEGATA	Field within the ACEE which points to GATE control blocks.
ADDSD	RACF TSO command used to define a dataset profile.
ADSP	Automatic DataSet Protection. At attribute that a user may have which ensures that datasets defined by the user are defined to RACF with discrete profiles automatically.
ALTLIB	TSO command used to alter the normal CLIST search sequence.
APPC	Application Program to Program Communication. A mechanism for providing communication between application programs.
ASXB	Address Space eXtension Block. An extension to the ASCB which resides within each address space.
ASXBSENV	Field within the ASXB which points to the ACEE.
CLIST	Command LIST. A series of TSO commands and control statements within a programming language, also called CLIST.
dataspace	An area of storage used to hold data only. Similar to an address space insofar as it may be byte addressed from 0 to 2Gb. May be accessed by multiple address spaces.
DFSMS	Data Facility / System Managed Storage. The data management component of an MVS system.
DS1IND40	A bit within the DSCB which indicates that a discrete profile exists for a dataset.
DSCB	DataSet Control Block. A control block held on dasd within a VTOC, which defines the existence of a dasd dataset.

GATE	Generic Access Table Entry. A control block containing descriptors and profile names for generic profiles held in storage.
GLOBAL profile	A RACF profile used to provide fast access to resources.
GPRF	In-storage generic profile map. Contains generic profiles held in storage.
HLQ	High-Level Qualifier.
initiator	A program used within an address space to select work from a queue. There are three types of initiator, JES, APPC and OMVS.
JES2	Job Entry Subsystem 2. An MVS component used to control batch jobs.
JESSPOOL	A RACF class used to protect spool datasets.
LSQA	Local System Queue Area. The high storage used within an address space below 16 megabytes.
OPERATIONS	Attribute of RACF user which allows access to most datasets.
OPERCMDS	RACF class used to control MVS operator commands.
RACFREE	TSO command supplied as part of RGM. Used to free GATEs and GPRFs in local storage.
RACLIST	A mechanism of bringing many profiles into storage to be accessed by one or many address spaces. (Also a RACF macro.)
RACROUTE	MVS SAF macro, normally used to invoke RACF services.
SDSF	Spool Display and Search Facility. Software used to display spool data on a JES2 spool.
SPECIAL	Attribute of RACF which allows access to any RACF command.
subpool	A storage designation. Subpools are used to define areas of storage which have like attributes. There are 255 subpools.
SYSPROC	DD name used for implicit invocation of CLISTs
VLF	Virtual Lookaside Facility. MVS component used to provide caching facilities for OS/390 components.

----------------------- End of Document -----------------------