You might think we recruited the FBI to appear on our latest cover of Hot Topics, but no—that’s Jay Brodfuehrer and Saheem Granados, two of IBM’s top designers and developers of security software. In their feature article “We’ve got your back(bone): TDS and RACF as the foundation of your security enterprise,” they introduce the theme of this issue: auditing and security for the enterprise.

And no, that’s not Fort Knox behind our resident tough guys Jay and Saheem, but it does illustrate how you can keep your assets safe and secure because—after all—your data is like gold to your business and requires the best in security software protection.

Security is indeed a hot topic for every enterprise these days, and the need for software that provides a seamless way to guard your systems and the most effective means to audit your data is paramount in today’s IT world.

And this issue has the articles that tie together security and auditing in today’s complex heterogeneous and distributed environments. Once you’ve read the overview on how RACF and TDS can provide that security foundation, take a look at how to ensure end-user accountability through z/OS ID propagation in an article by George Markouizos, Eric Rosenfeld, and Ira Ringel. For comprehensive auditing of your systems, be sure to read Richard Young’s article; it describes an effective way to let z/OS audit your Linux systems on System z. Then Bob Fairlie shows you how to use JZOS to access system logging of your Java applications.

Issue 22 is chock-a-block with advice and “how to” information from so many of our security gurus that you are certain to find the reading rewarding.

We might have streamlined this issue, but what we offer is pure gold for the system programmer. From articles that explore digitally-signed code to several on using Integrated Cryptographic System Facility (ICSF), including a best practices guide, this issue takes the headache out of planning your enterprise security and auditing.

And we haven’t forgotten our V1R11 users either with informative articles on InfiniBand links and FICON channel path management, JCL, SMB, and a great article on IBM’s test environment implementation of SAP Cloud computing. There’s another update on how the IBM Academic Initiative is addressing z/OS skills for future System z programmers and another installment of Ask Mr. Catalog, who answers all your questions about DFSMS catalog management.

Finally, be sure to unlock the clues of our very first crossword puzzle with an emphasis on this issue’s security and auditing theme. (Answers to be provided on our Hot Topics Web site.)

In our last issue we highlighted six simple words to describe the z/OS experience, so let me end with this:

“Safe, Simple, Secure systems with z/OS”

Oh! One final thing: I’d like to welcome Kristine Logan to the roster of z/OS Hot Topics managing editors. Welcome Kristine!
Contents:  Featured articles

4  TDS and RACF as the foundation of your security enterprise
   JAY BRODFUEHRER AND SAHEEM GRANADOS

7  use z/OS identity propagation to ensure end-user accountability
   GEORGE MARKOUIZOS, ERIC ROSENFELD, AND IRA RINGEL

9  digitally sign code on z/OS
   JUN OGATA, MICHAEL ONGHENA, AND SCOTT WOOLLEY

12  z/OS as the central audit repository for Linux on System z
    RICHARD YOUNG

15  Accessing System Logging and RAUDITX from a Java application using JZOS
    ROBERT FAIRLIE

6  zTidbitz

18  Best practices for securing ICSF resources
    BEN ROGERS, WILLIAM C. JOHNSTON, AND JONATHAN M. BARNEY

22  Have your cake and eat it too with protected-key CPACF
    STEVEN R. HART AND ERIC ROSSMAN

24  How ICSF keys are shared in a sysplex
    RONALD EDICK

27  Retrieving RACF information using REXX
    MIKE ONGHENA AND BRUCE WELLS

29  access your code using Java
    Some assembler required!
    ROBERT FAIRLIE

33  Ask Mr. Catalog
    STEPHEN BRANCH

34  Mail fast, quick, and easy with CSSMTP
    KIMBERLY BAILEY

35  zFavorites

36  Channel surfing: Give DCM the remote
    WILLIAM ROONEY, DALE RIEDY, AND STEPHEN KOCIK

38  Tailor your HCD profile using the new Profile Options dialog

39  Migrate to Parallel Sysplex InfiniBand coupling links
    FRANK HAGERTY, MIKE HRENCECIN, AND GEORGE NG

42  IBM System z education for a smarter planet
    KATHY PFEIFFER AND DON RESNIK

44  From the shell to JCL and back again
    JOHN FISCHER

45  Tuning up SMB
    KARL LAVO

47  DB2 for z/OS, the heart of a System z Test Cloud for SAP
    THOMAS H. GNECH, STEFFEN KOENIG, AND HOLGER J. SCHELLER

50  What can DEVSERV QL do for you? Probably more than you think
    BOB BURCHFIELD

52  Take control of storage management and message handling
    MEENAKSHI SRIPAL

53  Our contributors

54  Corrections and clarifications

55  Hot Topics Crossword

Back cover:

z/OS Communications Server gets social!
    FERNANDO BARSOBBA, SAM REYNOLDS, AND SPENCER SMITH
The IBM® Tivoli® Directory Server (TDS) for z/OS® and z/OS Security Server Resource Access Control Facility (RACF®) are two robust technologies that separately provide critical function and services within the enterprise. But did you know that you can combine the security strengths of TDS and RACF to offer the best in security, even in a heterogeneous IT environment?

Strengths of TDS and RACF

TDS is an implementation of the Lightweight Directory Adapter Protocol (LDAP) server based on numerous open LDAP protocol Request for Comments (RFCs) and Internet standards. Since its introduction in 1976, RACF has become the premier external security manager for securing the enterprise. Together, TDS and RACF work to bring the flexibility and assurance of open standards to the time-tested power and stability of RACF, providing the security foundation for the enterprise.

The relationship between TDS and RACF allows you to introduce RACF technology into enterprises that have adopted newer technologies but might not directly interact with RACF. Conversely, RACF installations can leverage Internet and LDAP protocol standards to retrieve and manage RACF specific data in more flexible ways. Most importantly, the synergy between TDS and RACF allows security officers the power, stability, and flexibility to implement security controls that reduce IT security risks.

The IT security environment today

Consider a typical IT environment today where various operating systems and hardware are deployed. For example, zLinux systems, IBM AIX® systems, and IBM z/OS systems can all be deployed to provide mission critical services to the enterprise. At the very least, security officers most often implement three crucial IT security controls to protect these systems:

- Authentication
- Authorization
- Logging of security events for future auditing.

This article covers TDS and RACF features that can be used to implement these three controls. We will discuss native and remote authentication, remote authorization, and remote security event logging. In addition, we will cover how to exploit TDS and RACF function to do things like remote RACF resource profile access and maintenance and RACF password synchronization across the enterprise or even across different security software.

Authentication

TDS and RACF separately provide authentication as a service. Although RACF and TDS allow for more complex authentication mechanisms like Kerberos, X.509 certificates, and pastickets, for the sake of this article, we will discuss basic non-administrator user and password authentication. (Also, in the case of TDS, you can use Secure Sockets Layer (SSL) to provide additional security controls over the flow of sensitive data.)

Using the LDAP protocol, authentication is performed by binding to the TDS server. In order to bind to a server, an LDAP entry must exist within the directory information tree (DIT) of the TDS server. This LDAP entry represents the entity that is being authenticated. The entry must consist of attributes, one of which must be a distinguished name (DN) attribute. A user password attribute can also exist. The entity being authenticated provides its DN and password. Because the authentication can be performed using TCP/IP communications, you can attempt authentication from any system within the enterprise. This permits a distributed RACF-based implementation of the authentication security control and does require some TDS configuration and the use of certain LDAP hierarchies, objects, attributes, and naming conventions.

Configuring SDBM

You can configure the TDS server to include a branch of the DIT that represents the RACF database, including RACF user profiles. The RACF branch is referred to as the SDBM backend in TDS.

Entries within SDBM have a standard DN naming convention. The LDAP DN of a RACF user profile contains the RACF user id as a component. When LDAP performs the bind using the SDBM DN of a user profile, RACF performs the authentication.
In addition to SDBM, you can add or update standard LDAP user entries in other branches of the DIT to contain a mapping from the LDAP DN of the entry to a RACF user ID. This mapping of the LDAP DN to a RACF user ID is referred to as native authentication in TDS. If TDS is configured correctly, mapped LDAP user binds allow RACF to perform the authentication using the mapped RACF user ID.

SDBM and native authentication that make use of TDS and RACF provide enterprise-wide authentication, even from within heterogeneous software and system environments. For example, Linux® systems that rely on pluggable authentication modules (PAM) to perform authentication can use an architected PAM module that communicates with TDS through TCP/IP and the LDAP protocol to perform an LDAP bind so that RACF performs the actual authentication.

Authorization and auditing
Authorization and logging of security events for auditing are two critical security controls. RACF provides services that implement these security controls. Remote access to these technologies allows security technologies throughout a heterogeneous software and system environment to leverage RACF in order to satisfy any of authorization and security event logging requirements.

Configuring ICTX
As with configuring TDS and the SDBM backend for authentication, you can configure TDS to include an ICTX backend for authorization or logging. This ICTX configuration allows TDS to process LDAP extended operations requests over TCP/IP for authorization and auditing using RACF.

Before issuing ICTX requests, you must ensure that a special LDAP bind is performed. As with the SDBM entry, you must use a DN that contains a RACF user ID. Also, the bind DN must contain a DN attribute that informs TDS to correctly authenticate with RACF.

When the LDAP bind succeeds, the remote entity can issue ICTX requests to perform authorization or to log security events for future auditing. You can then check for authorization of a RACF user ID or a RACF group. You can request the following levels of authority: READ, UPDATE, CONTROL, or ALTER. The resource being controlled must match a RACF profile and cannot be in the DATASET, USER, or GROUP general resource classes.

You also can use the ICTX backend to audit security logging of events. You can log type-80 system management facilities (SMF) records for access attempts based on user, class, resource, or any other supported criteria.

RACF management through TDS
In addition to TDS and RACF management of authentication, authorization, logging of security events through SMF, or resource protection, in your heterogeneous IT environment, you can use TDS to manage RACF.

Updating user passwords
One crucial piece of RACF data is a user’s password. Using the SDBM backend and LDAP, TDS allows RACF user passwords to be modified through TCP/IP. You can issue LDAP modify requests against TDS SDBM backend user profile entries. A user password modification of the LDAP entry results in a corresponding update to the password in RACF. Changes to pass phrases are supported in the same way as password changes.

Managing remote RACF data
In addition to user passwords, TDS and RACF can also be used to remotely manage RACF data. The TDS SDBM backend allows LDAP search commands to return LDAP entries that can represent USER, GROUP, and CONNECT profiles. In addition to retrieving data, use LDAP commands to add new USER, GROUP, and CONNECT profiles.

You can modify many segments for USER profiles using LDAP modify commands (for example, the UID and home directory fields in the OMVS segment.) Here are some of the RACF actions you can perform through LDAP commands:
• Add, modify, and delete RACF users, groups, and general resources. Data set resources are not supported.
• Add, modify, and delete user connections to groups.
• Add and remove users and groups in general resource access lists.
• Modify some SETROPTS options that affect classes (for example, RACLIST).
• Retrieve RACF information for users, groups, connections, general resources, and class options.
• Retrieve RACF user password and password phrase envelopes.

Change logging
Throughout this article, we have outlined the possibilities of using RACF as the ultimate decider for authentication. Some heterogeneous environments will undoubtedly have more than one authentication engine. In this case, synchronization of user, password, or both might be required. TDS provides a feature called change logging that can add special LDAP entries to the DIT when modifications of entries within the DIT occur. You can configure TDS and RACF so that any modifications to RACF passwords result in the addition of change
log entries to TDS. This logging allows for use of TCP/IP and LDAP. For example, IBM Tivoli Directory Integrator (TDI) can be deployed to await change log entries, and when it encounters those changes, it can synchronize the changes throughout the enterprise.

**Watch your back(bone)!**

TDS and RACF are two technologies that separately provide security and data management services to protect your IT environment. TDS implements LDAP, an RFC-based protocol, that manages data in a hierarchical directory tree structure. With the data management, LDAP operations like bind, search, add, and modify, and user-definable extended operations, allow security controls to be implemented and maintained through an open standard over TCP/IP networks. Because LDAP is an open standard, you can use it to implement common security controls throughout all kinds of operating environments.

RACF is a mature technology that has secured IBM mainframes since 1976. As an IBM mainframe resource manager, RACF implements System Authorization Facility (SAF) interfaces, but, by itself, RACF is not easily leveraged or managed from different systems within heterogeneous environments. Together, TDS and RACF combine the best of both worlds: open standards with the power of IBM mainframe security, providing the backbone of your installation security.

**Find out more:**


**zTidbitz**

Finally! A cheat sheet for z/OS!

Imagine, someone has finally boiled z/OS down for us on a Web site! The zTidbitz Web site contains a collection of condensed and summary information on a wide range of z/OS and System z® hardware topics. Use them as cheat sheets or reference sheets because each one condenses a ton of product documentation into a short topic. So, if you need a quick reminder for a topic you’re familiar with, you’ll find these cheat sheets very helpful.

Be sure and visit our Web site: [www.idcp.org/ztidbitz](http://www.idcp.org/ztidbitz)

Here’s a sampling:

- Basic z/OS features
- Java™
- Operator commands
- Performance monitoring
- System availability
- Server hardware
- Total cost of ownership
- And much much more!
In today's distributed environments, end users are known by their Internet or distributed user identities, but the z/OS Security Server deals only with RACF user IDs. As a transaction flows from the z/OS subsystems (such as CICS® and IMS™) to RACF, an application may associate all users with a single shared RACF identity. Some applications do this to avoid having to force every user to authenticate, but this approach can compromise end-user accountability.

To address this problem, RACF has, in z/OS V1R11, introduced an identity propagation function. z/OS identity propagation makes the identity of the end user securely available to the back-end business logic program and transaction processing z/OS subsystems, at the application and security-domain level. Let’s take a look at how this new capability can help ensure end-user accountability.

A world without z/OS identity propagation

Without z/OS identity propagation, there is no mechanism to pass the distributed user identity information to RACF. The distributed user information is lost along the way, and only the RACF user ID is passed on to the mainframe applications. As a result, the distributed identity of the end user who initiated the transaction plus other security-relevant information is not available for RACF to include in SMF audit records written during the transaction. Figure 1 shows such a world.

A world with z/OS identity propagation

z/OS identity propagation includes function to consistently map distributed user identities into RACF user IDs by way of the RACMAP command, supporting one-to-one as well as many-to-few and many-to-one scenarios. It also includes a high-performance pass-by-reference mechanism that will exploit sysplex communications using the R_cacheserv callable service. This exploitation of the cross-system coupling facility (XCF) enables z/OS identity propagation to pass detailed user security information among various z/OS subsystem components, while minimizing contention on the z/OS security registry (the RACF database). The data areas that contain this information are the Distributed Identity Data Structure (IDID) and the Identity Context Reference Extended (ICRX) data areas. Figure 2 illustrates how z/OS identity propagation works.
The following steps detail how an application would take advantage of z/OS identity propagation. If RACF is to derive user IDs, the security administrator can first use the RACMAP command to define distributed identity filters.

1. The application authenticates the distributed identity.

2. The application builds an IDID with a Distinguished Name (DN) and registry.

3. The application calls the RACROUTE REQUEST=VERIFY macro or the initACEE service to create an accessor environment element (ACEE). The ACEE is a control block that defines the runtime security environment.

   - If the user ID is not being derived by RACF, the application specifies the USERID and IDID parameters when calling the RACROUTE REQUEST=VERIFY macro or the initACEE service.
   - If the user ID is being derived by RACF, the application calls the RACROUTE REQUEST=VERIFY macro (using the ICRX parameter to identify where the application has built the ICRX with an IDID containing the DN and registry), or else calls the initACEE service specifying only the IDID parameter.

4. The application calls the R_cacheserv service, specifying the store function (function code 7, option 1) to get an ICRX. If the application is running in supervisor state and has specified the ACEE as a parameter on the R_cacheserv store call, that ACEE will be cached. Otherwise, the effective ACEE, which is pointed to either by the task control block (TCB) or the address space extension block (ASXB), will be cached. The output of this operation is a complete ICRX that can move with the unit of work through the system or sysplex. The complete ICRX contains the Identity Context Reference (ICR) and the IDID with section two completed.

5. When the security environment has to be recreated, the application issues RACROUTE REQUEST=VERIFY with the ICRX that was created in step 4. If the security environment does not need to be recreated, the application can clean up the cache by calling the R_cacheserv service, specifying the remove function (function code 7, option 3). The application would pass the ICRX created in step 4 to the R_cacheserv service. The ICRX times out after one hour and will be removed at the next cache cleanup.
The use of digitally signed code can increase the reliability and security of a system by adding controls on executable programs running on that system. z/OS V1R11 Security Server RACF gives you the ability to sign modules cryptographically and to require certain programs to have a digital signature before they can run.

A digitally signed program consists of two parts—the executable code and a digital signature. The digital signature is the result of a cryptographic operation between the executable code and the private key of an asymmetric key pair. When z/OS loads a signed program, it validates the signature using the private key's corresponding public key.

If z/OS detects a problem during validation, it does not load the program. There are several reasons a program might fail the validation step. It is possible that the program was tampered with or is otherwise corrupt. It is also possible that you tried to load an unsigned version of a program when the system configuration requires a signature. This latter case might indicate an accidental violation of change-control procedures. For example, a test version of the code may have been erroneously installed in a production environment.

**Program signing setup**

Only program objects residing in a partitioned data set extended (PDSE) can be signed.

z/OS cannot sign load modules stored in a partitioned data set (PDS), or programs stored in a z/OS UNIX® System Services (z/OS UNIX) file system. However, z/OS UNIX programs can link to signed executables in PDSE data sets. The z/OS binder requires some configuration to sign your programs. Here is an overview of the setup steps your security administrator must perform:

1. Create a certificate chain (from 1 to 10 certificates) to use in the signing process. To do this, the system administrator first creates the top-level certificate authority (CA) certificate.

   ```plaintext
   RACDCERT CERTAUTH GENCERT SUBJECTSDN
   (OU('XYZ Code Signing CA') O('XYZ Corp.') C('US'))
   SIZE(2048) WITHLABEL ('XYZ Code Signing CA')
   ```

   Then, the system administrator defines the certificate chain. Here, we create a code-signing certificate, signed by the CA certificate.

   ```plaintext
   RACDCERT ID(XYZBLD) GENCERT SUBJECTSDN(OU('XYZ Code Signing Cert.')
   O('XYZ Corp.') C('US')) SIZE(1024) WITHLABEL('XYZ Code Signing')
   SIGNWITH(CERTAUTH LABEL('XYZ Code Signing CA'))
   KEYUSAGE(HANDSHAKE DOCSIGN DATAENCRYPT)
   ```

   Although, in this example, we have generated certificates using the RACDCERT GENCERT command, note that you can also use third-party certificates that have been imported into the RACF key store using the RACDCERT ADD command.

2. Create a key ring for all the certificates in the certificate chain. The name of key ring must be all upper case.

   ```plaintext
   RACDCERT ID(XYZBLD) ADDRING(CODESIGNKEYRING)
   ```

3. Connect the CA certificate to the key ring.

   ```plaintext
   RACDCERT ID(XYZBLD) CONNECT(RING(CODESIGNKEYRING)
   LABEL('XYZ Code Signing CA'))
   ```

4. Connect the code signing certificate to the key ring.

   ```plaintext
   RACDCERT ID(XYZBLD) CONNECT(RING(CODESIGNKEYRING)
   LABEL('XYZ Code Signing Cert.') DEFAULT)
   ```

5. Create a FACILITY class profile to identify the token ring to use for program signing. The following profile will be used for all programs that are to be signed, but you can define additional granularity based on user and group.

   ```plaintext
   RDEFINE FACILITY IRR.PROGRAM.SIGNING APPLDATA
   ('SHA256 XYZBLD/CODESIGNKEYRING')
   ```

6. Make sure that the user who is signing code has the authority to read the keys in the key ring certificates.

   ```plaintext
   PERMIT IRR.DIGTCERT.LISTRING ID(XYZBLD) ACCESS(READ) CLASS(FACILITY)
   ```
Let’s sign some code

You can sign programs using the z/OS binder by specifying the new sign option as a binder parameter. You can use this new option:

- On the z/OS UNIX compile command:

  ```bash
  # c89 -Wl,sign -o "//MY.LOADLIB.PDSE(MYPROG14)" myprog14.o
  ```

- In JCL:

  ```
  //LINKEDIT JOB
  //STEP1 EXEC PGM=IEWL,REGION=1024K,PARM='SIGN,AMODE=31'
  //SYSPRINT DD SYSOUT=*  
  //OBJECT DD DSN=SOURCE.OBJ,DISP=SHR 
  //SYSLMOD DD DSN=MY.LOADLIB.PDSE,DISP=SHR 
  //SYSLIN DD * 
  INCLUDE OBJECT(MYPROG14) 
  ENTRY MYPROG14 
  NAME MYPROG14(R) 
  /*
  ```

If there is a problem with your code-signing configuration, the binder issues error messages with RACF return codes. For more information on these return codes, see the references at the end of this article. Here are some common reasons for errors:

- The APPLDATA field of the IRR.PROGRAM.SIGNING profile is an upper-case only field. Any text placed into the APPLDATA field is automatically folded to uppercase by RACF. Any mixed-case key ring name will not be found.
- The syntax of the APPLDATA field in the IRR.PROGRAM.SIGNING profile is very strict. Ensure you correctly specify the signing algorithm, owner, and key ring name.
- All of the certificates in the certificate chain must be in the key ring, and the user signing the code must have access to read them. This includes the top CA certificate, the code-signing certificate, and all certificates (if any) in between.
- The code-signing certificate must have the appropriate KEYUSAGE attributes of HANDSHAKE DOCSIGN DATAENCRYPT. The code-signing certificate must be marked as the DEFAULT certificate in the key ring. It must also have a private key.

Program verification setup

Once linked into a PDSE on your target system, your first signed program should be executable if there are no restrictions in place for it. z/OS ignores the program signature information until the RACF administrator performs the necessary configuration to activate program verification.

The RACF administrator must first perform the one-time setup required to enable the RACF program verification module. The setup includes (but is not limited to) the following two steps:

- Save the top-level CA certificate for other systems to use in the verification process. Only save the root certificate. The other certificates in the signing chain are stored with the signature.

```bash
RACDCERT EXPORT (LABEL('XYZ Code Signing CA')
CERTAUTH DSN('MYSIGN.CA.CERT.DSN')
```
1. Create the key ring that connects the root signing CA certificates for the system. This key ring should be a different from the key ring you use for signing the code.

   ```
   RACDCERT ID(RACFADM) ADDRING(SYSTEM.CODE.VERIFY.KEYRING)
   ```

2. Define and activate the FACILITY class profile IRR.PROGRAM.SIGNATURE. VERIFICATION that specifies the name of that key ring.

   ```
   RDEFINE FACILITY IRR.PROGRAM.SIGNATURE.VERIFICATION APPLDATA ('RACFADM/SYSTEM.CODE.VERIFY.KEYRING ') 
   SETROPTS RACLIST(FACILITY) REFRESH 
   ```

**Program verification steps**

The RACF administrator must perform the following steps for each signed program that needs verification:

1. Add the root CA certificate of the certificate originally used to sign the code as a trusted CA (if it is not already in the RACF database), and connect that root CA certificate to the verification key ring.

   ```
   RACDCERT CERTAUTH ADD(MYSIGN.CA.CERT.DSN) 
   WITHLABEL('XYZ Code Signing CA') TRUST 
   RACDCERT ID(RACFADM) CONNECT(CERTAUTH LABEL('XYZ Code Signing CA') 
   RING(SYSTEM.CODE.VERIFY.KEYRING))
   ```

2. Create and activate the PROGRAM class profile by specifying the SIGVER options for it.

   ```
   RDEFINE PROGRAM MYPROG14 ADDMEM('MY.LOADLIB.PDSE '//NOPADCHK) 
   SIGVER(SIGREQUIRED(YES) FAILLOAD(ANYBAD) SIGAUDIT(ANYBAD))UACC(READ) 
   SETROPTS WHEN(PROGRAM) REFRESH
   ```

Although, in this example, we have created a discrete profile, note that you could also create a generic profile to cover multiple programs.

**Program verification options**

The SIGVER options in the PROGRAM class profile tell z/OS what to do when loading a signed program.

- **SIGREQUIRED(YES|NO)** specifies whether the program module must include a digital signature. If YES, RACF verifies the signature to ensure the module is unchanged and the code signer(s) are trusted, returning a success or failure result. If NO, the signature is optional, but RACF does verify it if it is present.
- **FAILLOAD(ANYBAD|BADSIGNONLY|NEVER)** specifies whether to allow the program to load for execution in the event of a verification failure. When you specify ANYBAD, the program is not loaded for any type of verification failure. When you specify BADSIGNONLY, the program can load if the failure resulted from an untrusted signer, but the data signature is correct. If you specify NEVER, the program is always allowed to load.
- **SIGAUDIT(ALL|SUCCESS|ANYBAD|NONE)** specifies whether to log signature verification as audit events. When you specify ALL, RACF logs both success and failure signature verifications. When you specify SUCCESS, RACF only logs successful verifications. When you specify ANYBAD, RACF logs all verification failures while BADSIGNONLY will not log for untrusted signer-only conditions. When you specify NONE, the system performs no verification logging.

**Program verification messages**

RACF issues console messages for verification failures that are recorded as specified by the SIGAUDIT option. The messages are important in problem diagnosis. For example:

**ICH440I Program signature error retecode/rsncode for program program-name in library library-name. The program was not loaded.**

The system issues this message when a signature verification failure prevents a program from being loaded. An associated message follows to describe the reason for the failure, such as a missing signature or untrusted certificate signer(s). If the FAILLOAD option allows the program to be loaded in spite of a verification failure, the system issues message ICH441I instead of ICH440I.

**References**

The return and reason code values included in the program verification error messages reflect codes returned from the VERFINAL function of the R_PgmSignVer callable service. Return and reason code values displayed in binder error messages during program signing are returned from the SIGINIT and SIGFINAL functions of R_PgmSignVer. See z/OS Security Server RACF Callable Services, SA22-7691, for explanations of these codes.

Keep all your “Penguins” in a row with z/OS

z/OS as the central audit repository for Linux on System z

BY RICHARD YOUNG

You’re probably aware that the Linux audit system produces audit records for events within the Linux operating system. Did you know that you can configure which events you would like to audit and where the audit records are sent? For example, you can audit accesses to files or to Linux system calls. The audit dispatcher distributes the audit records to different destinations through the different plug-ins that have been configured. Figure 1 provides an overview of this audit distribution by the Linux audit system.

Linux and System z

When you consolidate Linux servers in a System z environment, you often want to report on what all of them are doing. You might even be required to audit all of your servers for regulatory compliance reasons. It is also likely that you have existing audit and reporting processes that have been in place in your z/OS environment for many years. So there is often value in being able to leverage these existing tools, processes, and staff.

The good news is that z/OS can be used as a central repository for audit records created by the Linux kernel audit system. This is accomplished through the audit dispatcher and the z/OS remote plug-in that communicates with the Lightweight Directory Access Protocol (LDAP) server. As a result, you can leverage existing z/OS system management facilities (SMF) reporting procedures and tools for your Linux environments on multiple hardware platforms. Figure 2 provides an overview of this auditing process for z/OS.

Figure 1. Linux audit system overview

Figure 2. Linux auditing in a z/OS environment
Using TDS LDAP and SMF records for auditing

To begin capturing Linux audit records to z/OS SMF, you need to have an active IBM Tivoli Directory Server (TDS) LDAP server. To use the required ICTX extended operations backend, you must ensure that the instance of TDS LDAP that captures your audit records runs in 31-bit and not in 64-bit mode. To add the backend to LDAP include the following in your ds.conf file:

```
# ICTX extended operations support
section database ictx ITYBIC31
suffix "cn=ictx"
```

In order to make the ITYBIC31 module available for auditing, you must add /usr/lib to the LIBPATH in the ds.ennvars file.

Defining RACF permissions

Next, you need to define several z/OS Security Server Resource Access Control Facility (RACF) resources and permissions or use a similar security product. First, use the RACF RDEFINE command to permit the LDAP server access to process remote audit requests:

```
RDEFINE FACILITY IRR.RAUDITX UACC(NONE)
PERMIT IRR.RAUDITX CLASS(FACILITY) ID(LDAPSRV) ACCESS(READ)
```

To record all audit events for the @LINUX class, use the following commands:

```
RDEFINE @LINUX * UACC(NONE) AUDIT(ALL(READ))
SETR RACLIST(@LINUX) REFRESH
```

To avoid the LDAP server issuing an error during start be sure to permit the LDAP server to write to an identity cache as follows:

```
RDEFINE FACILITY IRR.RCACHESERV.ICTX UACC(NONE)
PERMIT IRR.RCACHESERV.ICTX CLASS(FACILITY) ID(LDAPSRV) ACCESS(UPDATE)
SETR RACLIST(FACILITY) REFRESH
```

The good news is that z/OS can be used as a central repository for audit records created by the Linux kernel audit system.

Use the following RACF commands to define the @LINUX CDT class:

```
REDEFINE CDT @LINUX CDTINFO(POSIT(493) +
    FIRST(ALPHA,NATIONAL,NUMERIC,SPECIAL) +
    OTHER(ALPHA,NATIONAL,NUMERIC,SPECIAL) +
    RACLIST(REQUIRED) CASE(ASIS) +
    GENERIC(ALLOWED) DEFAULTUACC(NONE) MAXLENGTH(246))

SETR CLASSACT(CDT)
SETR RACLIST(CDT)
SETR RACLIST(CDT) REFRESH
SETR CLASSACT(@LINUX)
SETR RACLIST(@LINUX)
SETR GENERIC(@LINUX)
```
Starting the Linux configuration
To ensure that the required audit rpms are installed, issue the following from the Linux server:

```
rpm -qa | grep audit
```

Check for the following packages:
- `audit`
- `audit-audispd-plugins`

Updating /etc/audispd/audispd-zos/remote
Next, update the `/etc/audispd/audispd-zos/remote` file:

```
server = 10.9.8.7
port = 389
user = LINXAUDT
RACF
password = yourpass
```

Validating /etc/audispd/plugins.d/audispd-zos-remote.conf
Now validate that the `/etc/audispd/plugins.d/audispd-zos-remote.conf` file has the "path" line configured correctly. For example, in my configuration, the default value was incorrect (`/sbin/audispd-zos-remote`), so I had to modify the path as `/usr/sbin/audispd-zos-remote`.

At this point, ensure that definitions for `/etc/audit/audit.rules` have been customized for your business requirements.

Restarting auditd
Now, restart the `auditd` process by issuing the following command:

```
rcauditd restart
```

Check `/var/log/messages` for errors. If everything is successful, you can expect messages similar to the following:

```
audispd-zos-remote: pid=25883: starting with pid=25883
audispd-zos-remote: pid=25883: Using configuration file: /etc/audisp/zos-remote.conf
audispd: af_unix plugin initialized
audispd: audispd initialized with q_depth=64 and 2 active plugins
auditd[25880]: Started dispatcher: /sbin/audispd pid: 25882
```

Check to see if your “penguins” are all in a row!
At this point, you could trigger some audit events by viewing or updating an audited file on Linux. z/OS writes the audit records to SMF record type 83 subtype 4. To dump the SMF data sets and view the records, use the RACF Database Unload utility (IRRDBU00) in XML format or a similar tool.

There you have it! A way to centralize your z/OS Linux auditing and keep it in good order!

For information about SMF, see z/OS MVS System Management Facilities (SMF), SA22-7630.
For information about RACF commands, see z/OS Security Server RACF Command Language Reference, SA22-7687.
Are you ready for the AUDIT? JZOS is!

Accessing system logging and RAUDITX from a Java application using JZOS

BY ROBERT FAIRLIE

You’re ready to create a new business application, and you know you need to create a trail for both audit readiness. If you get it right, the Business Controls police may even smile... nicely. Not that smile that gives you cold sweats when they say “prove it!”

JZOS can help you get on the audit trail. The IBM JZOS Batch Toolkit for z/OS SDKs is a set of tools that enhances Java batch capabilities and use of system interfaces on z/OS. In this article, I’ll give you a high level view of using JZOS to access system logger and RAUDITX resources from a Java application.

System logger

As part of z/OS Java, JZOS provides access to system logger using the JZOS class, ZLogstream. ZLogstream lets you use macros to connect with (IXGCONN) and write to (IXGWRITE) system logger. For example, working with a logstream can help you find information about who accessed or updated key files.

Figure 1 shows a JCL sample defining a system logger data set.

```java
import com.ibm.jzos.ZLogstream;
import com.ibm.jzos.ZLogstreamException;

try {
    String logName = "hlq.mylog.dataset";
    String mylogrecord = "Test log stream record write";
    byte[] bytes = new byte[200];
    ZLogstream zlogger = new Zlogstream(logName);
    zlogger.setWriteRetryCount(100);
    zlogger.write(bytes, 0, bytes.length, false);
    zlogger.close();
} catch (ZLogstreamException EX) {
    System.err.println("System logger error message <" +
                        EX.getMessage() + ">");
    System.err.println("System logger ReturnCode/ReasonCode <" +
                        EX.getReturnCode() + "/" +
                        EX.getReasonCode() + ">");
    System.err.println("System logger resource <" +
                        EX.getLogstreamName() + ">");
    EX.printStackTrace();
    return;
}
```

Figure 2 shows how easy it is to access system logger. It also shows how to retrieve error information from system logger in the event of a connect or write failure. You can run this code from either an OMVS command line or from the JZOS batch launcher.
Here are a few hints for using system logger from your Java application:

- When designing a Java application that accesses system logger, keep in mind that a DASD-only log stream can slow the performance of your application.
- Coding the IXGWRITE with macro with the ASYNCONRESPONSE parameter will speed things along in your application. However, because the caller is not informed when the request completes, your application may miss a failure for that write.
- You must write your own browse tool using the IXGBRWSE macro to access log records. If your application is creating text-only records, you can also use the WebSphere Application Server for z/OS log browse utility (BBORBLOG) to read the records later.

Figure 3 shows a sample of code that calls BBORBLOG to read your system logger data set:

```
/* REXX */
/* Browse a z/OS Logstream using BBORBLOG */
trace o
/* Customize the following line to point to your */
/* location's SBBOLOAD library. */
/* environment('STEPLIB', 'SYS1.SBBOLOAD')
parse arg logstrm format .
/* if arg logstream is blank, exit */
if logstrm = '' then exit(1)
/* if arg lformat is blank, default to 80 */
if format = '' then format = "80"
/* Create a filename with your userid as part of */
/* the output filename */
qual = userid()
file_name = "/tmp/" || qual || ".errorlog"
/* Create the output file in /tmp directory */
"rm" """ || file_name
"touch" """ || file_name
/* Allocate file and call BBORBLOG */
call syscalls 'SIGOFF'
call bpxwdyn "alloc fi(bbolog) path('' || file_name || '')" address LINKMVS "BBORBLOG logstrm format"
/* Release allocated file */
call bpxwdyn "free fi(bbolog)"
exit(0)
```

Figure 3. Using BBORBLOG to read a system logger data set

**RAUDITX**

As part of the z/OS Java offering, JZOS provides an interface for writing SMF 83 audit records using the r_audix Callable services, IRRSAX00 and IRRSAX64.

To use this these services, the calling user ID must have READ authority to the IRR.RAUDITX profile in the FACILITY class. See z/OS Security Server RACF Callable Services, SA22-7691, for more information on writing SMF 83 audit records.

Figure 4 shows how easy it is to create and write an SMF 83 audit record, and how to retrieve information from RACF/SAF in the event of failure.
import com.ibm.jzos.Rauditx;
import com.ibm.jzos.RauditxException;

try {
    Rauditx rauditx = new Rauditx();
    /* Set event parameters */
    rauditx.setEventFailure();
    rauditx.setAuthenticationEvent();
    rauditx.setEvent(100);
    rauditx.setQualifier(25);

    /* Set component name and FMID */
    rauditx.setComponent("ABCD");
    rauditx.setFmid("ABCD123");

    /* Set component assigned subtype */
    rauditx.setSubtype(9);

    /* Setup error message to console in case of failure */
    rauditx.addMessageSegment("ABCD123 CONSOLE LINE 1");
    rauditx.addMessageSegment("CONSOLE LINE 2");

    /* Add relocate sections to Rauditx record written to SMF */
    rauditx.addRelocateSection(100, "MyUser");
    rauditx.addRelocateSection(101, "MyBindUser");

    /* This sample can continue on with addRelocateSections */
    /* but why belabor the point. */

    /* Write an Rauditx record */
    rauditx.issue();
}

} catch (RauditxException EX) {
    /* Print out messages and return codes from RACF */
    System.err.println("RACF error message " +
    EX.getmessage() + ">");
    System.err.println("RACF ReturnCode/ReasonCode " +
    EX.RacfgetReturnCode() + "/" +
    EX.RacfgetReasonCode() + ">");
    System.err.println("SAF ReturnCode " +
    EX.SafgetReturnCode() + ">");

    /* Print out Java stack trace */
    EX.printStackTrace();
    return;
}

Figure 4. Writing an Rauditx record to SMF and retrieving information from RACF/SAF

Hint: If you use the RAUDITX class in your application, your installation must either create a new, or modify the current, SMF record extract tool to select those records pertinent to your application.

Final Word

By implementing the RAUDITX and zLogstream classes within your Java application, you can tap into the strength of z/OS for creating a secure audit trail. As an application developer, you must work with your security administrator to ensure that the application has the proper authorization to access these system resources.

Keep in mind that you can’t trace everything. So, as you write your application, choose which records you want to trace.

For specific information on using these two JZOS classes, see the JZOS Java information at the following Web site:

ibm.com/developerworks/java/zos/javadoc/jzos/index.html

New Java application

- Audit trail
- Recovery log
- Lunch with business control group
This article lays out specific examples for defining a role-based access control in Integrated Cryptographic Service Facility (ICSF) including key generation, key custody, and key use as well as roles for securing the functions surrounding the ICSF master key ceremony. To ensure appropriate use of each role, we’ll also review audit considerations. The three general categories of roles to consider when securing cryptographic resources within z/OS are administrators, key custodians, and key users.

- Key administrators, referred to as @CSFADM, focus on administering the ICSF address space and associated processes. A z/OS system administrator rather than a security administrator typically holds this role.
- Key custodians associated with ICSF include master key part custodians (@MKCUST), symmetric data key custodians (@SDKCUS), and asymmetric data key custodians (@ADKCUS).
- Key consumers are typically applications, but might also be individuals. We’ll describe roles associated with each consumer the best practices for securing each role.

The tables accompanying each role list the profiles accessible to the role whether the role is suitable for auditing. We assume read access to resource profiles unless we mention otherwise. In the tables any profile marked “Yes” has an audit log setting of all-read. Any profile left blank indicates no auditing of this resource is required.

### Roles associated with ICSF resources

#### ICSF Administrator (@CSFADM)

The ICSF Administrator role grants users the ability to start and stop cryptographic services, to re-encipher data sets such as the cryptographic key data set (CKDS) or the public key data set (PKDS), and to set master keys. This role performs the majority of the tasks to prepare the environment for the master key part owners during a master key change ceremony. This role also enables the changes made by the master key part owners. The ICSF administrator is typically not a master key or data key custodian, but performs the following tasks:

- Accesses administrative panels
- Modifies the ICSF options data set
- Adjusts cryptographic domains using the support element (SE) panel
- Manipulates the ICSF started tasks.

The ICSF administrator (@CSFADM) has read access to the CSFPMCI profile to set up ICSF, but you must remove access from this profile after ICSF is in production.

<table>
<thead>
<tr>
<th>Profile</th>
<th>Audit</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSFCMK</td>
<td>Yes</td>
</tr>
<tr>
<td>CSFPKDR</td>
<td>Yes</td>
</tr>
<tr>
<td>CSFREFR</td>
<td>Yes</td>
</tr>
<tr>
<td>CSFRENC</td>
<td>Yes</td>
</tr>
<tr>
<td>CSFSMK</td>
<td></td>
</tr>
<tr>
<td>CSFRSWS</td>
<td></td>
</tr>
<tr>
<td>CSFSSWS</td>
<td></td>
</tr>
<tr>
<td>CSFIQF</td>
<td></td>
</tr>
<tr>
<td>CSFPCAD</td>
<td></td>
</tr>
</tbody>
</table>
Master key custodian (@MKCUST)
In order to split the knowledge of the master key into multiple parts, the recommendation is to have a minimum of two master key custodians. Each custodian generates and uses one master key part for each data set (CKDS and PKDS). The best practice is at least one backup user for each custodian with access to the key material. The master key custodian typically has the following characteristics:
- Two or more people hold this role
- The master key custodian is not the ICSF administrator
- The master key custodian generates random numbers and enters key parts.

Data key custodian (@DKCUS, @SDKCUS, @ADKCUS)
Two security manager classes control access to key material in ICSF using:
- CSFSERV to limit access to ICSF callable services
- CSFKEYS to protect keys based on label name.

The controls for data key custodians (@DKCUS) are dependent on the symmetric and asymmetric custodians’ keys. The symmetric custodian and asymmetric custodian each have specific access requirements to CSFSERV class profiles.

Symmetric key custodian
Custodians of symmetric keys (@SDKCUS) require access to profiles that control key generation, export, import, and other services to manipulate DES, triple DES, and AES keys.

Asymmetric keys custodian
Custodians of asymmetric keys (@ADKCUS) require access to profiles that control RSA key pair generation, public key operations, PKDS alterations, random number generation, and other services to manipulate RSA keys.
Data key users
It’s important to consider the roles granted to applications, developers and testers as well as the naming convention you use for keys.

Applications
You can divide application functions into four categories:
- Symmetric (@SCRYP)
- Asymmetric (@ASCRYP)
- Hashing (@HCRYP)
- Financial (@FCRYP).

It is sometimes appropriate to grant applications or users access to any combination of these groupings providing your installation maintains separation of duties. In production environments, users or groups connected to application groups must not also have connections to any of the custodian groups. Applications need access to profiles in the CSFKEYS class to use key material in the CKDS, PKDS, and token data set (TKDS).

Users of symmetric key functions (@SCRYP) typically need the following callable services:
- Translation functions
- Encipher and decipher services
- Random number generation.

Symmetric (@SCRYP)
<table>
<thead>
<tr>
<th>Profile</th>
<th>Audit</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSFCTT</td>
<td></td>
</tr>
<tr>
<td>CSFCTT1</td>
<td></td>
</tr>
<tr>
<td>CSFDEC</td>
<td></td>
</tr>
<tr>
<td>CSFDEC1</td>
<td></td>
</tr>
<tr>
<td>CSFDCC</td>
<td></td>
</tr>
<tr>
<td>CSFENC</td>
<td></td>
</tr>
<tr>
<td>CSFENC1</td>
<td></td>
</tr>
<tr>
<td>CSFECO</td>
<td></td>
</tr>
<tr>
<td>CSFRNG</td>
<td></td>
</tr>
</tbody>
</table>

Users of asymmetric key functions (@ASCRYP) typically need the following callable services:
- Signature functions
- Encryption and decryption services
- Random number generation.

Asymmetric (@ASCRYP)
<table>
<thead>
<tr>
<th>Profile</th>
<th>Audit</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSFDAG</td>
<td></td>
</tr>
<tr>
<td>CSFDSV</td>
<td></td>
</tr>
<tr>
<td>CSFPKD</td>
<td></td>
</tr>
<tr>
<td>CSFPKE</td>
<td></td>
</tr>
<tr>
<td>CSFRNG</td>
<td></td>
</tr>
</tbody>
</table>

Hashing functions (@HCRYP) typically need the following callable services:
- Message Authentication Codes (MAC) and message digest functions
- Hashing services
- Random number generation.

Hashing (@HCRYP)
<table>
<thead>
<tr>
<th>Profile</th>
<th>Audit</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSFMGN</td>
<td></td>
</tr>
<tr>
<td>CSFMGN1</td>
<td></td>
</tr>
<tr>
<td>CSFMDG</td>
<td></td>
</tr>
<tr>
<td>CSFMDG1</td>
<td></td>
</tr>
<tr>
<td>CSFSWH</td>
<td></td>
</tr>
<tr>
<td>CSFSWH1</td>
<td></td>
</tr>
<tr>
<td>CSFRNG</td>
<td></td>
</tr>
</tbody>
</table>

Financial functions (@FCRYP) typically need the following callable services:
- Personal Identification Number (PIN) services
- Credit card validation services.

Financial (@FCRYP)
<table>
<thead>
<tr>
<th>Profile</th>
<th>Audit</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSNBCPE</td>
<td></td>
</tr>
<tr>
<td>CSNBPGN</td>
<td></td>
</tr>
<tr>
<td>CSNBCPA</td>
<td></td>
</tr>
<tr>
<td>CSNBPGP</td>
<td></td>
</tr>
<tr>
<td>CSNBPRTR</td>
<td></td>
</tr>
<tr>
<td>CSNBPRVR</td>
<td></td>
</tr>
<tr>
<td>CSNBPCU</td>
<td></td>
</tr>
<tr>
<td>CSNBPKY</td>
<td></td>
</tr>
<tr>
<td>CSNBSN</td>
<td></td>
</tr>
<tr>
<td>CSNDBC</td>
<td></td>
</tr>
<tr>
<td>CSNDBC7</td>
<td></td>
</tr>
<tr>
<td>CSNBCSV</td>
<td></td>
</tr>
</tbody>
</table>

Key-naming conventions
Users need access to services in order to perform cryptographic operations; however, this access is fruitless without access to key material. Implementing a strong key-naming convention can provide similar benefits to implementing role-based access control for ICSF services. Good key-naming conventions achieve the following goals:
- Ease of protecting key material using naming conventions that are unambiguous and allow generic or discrete protection profiles. Generic naming profiles are typically preferred.
- Ease of determining key:
  - Usage: Who uses it? Where is it used? What does it protect?
  - Life: When is it used? When is it created? When is it deleted?
  - Custodianship: Who owns it? Who creates it? Who deletes it?

These questions serve as the basis for determining a naming convention. It’s not always possible or appropriate to address all of the “W” questions in a naming convention. Table 1 pulls all these questions together.
Table 1. Key management conventions

The environment in which a key lives is usually the most obvious indication of its value. It’s often important to limit access to key material by using profiles that differentiate between logical environments like production and test, or application A and application B. Table 2 shows examples of a simple naming convention that places the environment as the first qualifier, such as:

<Environment>.<Application>.<Expiration>

Table 2. Environment naming convention

<table>
<thead>
<tr>
<th>Protection profile</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROD.EKM.*</td>
<td>Protects access to all keys associated with the EKM application in the PROD environment</td>
</tr>
<tr>
<td>PROD.EKM.X20100101</td>
<td>Discrete profiles can be used to restrict access to keys that are no longer used instead of deleting key material</td>
</tr>
<tr>
<td>PROD.*</td>
<td>A backstop for all keys in the PROD environment</td>
</tr>
<tr>
<td>TEST.*</td>
<td>A backstop for all keys in the TEST environment</td>
</tr>
</tbody>
</table>

Heretofore, we provide an all or nothing approach to key label access control. Regardless of role, the user has authority to create and delete keys if the roles called for any level of key usage.

In HCR7751, ICSF introduced granular key label access control in order to limit access to key material by label name. With this control, it’s possible to grant an application access to use a key, but deny its ability to modify the same key. Table 3 gives a brief overview of suggested access levels for various roles outlined above.

Role up
A well thought out role-based access control structure and key label naming convention can help provide improved security to your infrastructure. Many current regulations such as Payment Card Industry Data Security Standard (PCI-DSS), Health Insurance Portability and Accountability Act (HIPAA), and Sarbanes-Oxley require separation of duties. The intent of showing the roles and audit settings is to help show you how to provide a solid foundation for implementing such conventions.

For more information, see the following publications:
• z/OS V1R11.0 Cryptographic Services ICSF Administrator’s Guide, SA22-7521-13
• z/OS V1R9.0-V1R10.0 Cryptographic Services ICSF Administrator’s Guide, SA22-7521-12.

Table 3. Granular key label access control

<table>
<thead>
<tr>
<th>Function</th>
<th>Without granular key label access control</th>
<th>With granular key label access control</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read a key label</td>
<td>Read</td>
<td>Read</td>
<td>@SCRYP</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>@ASCROP</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>@HCRYP</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>@CRYPAPP</td>
</tr>
<tr>
<td>Create a key label</td>
<td>Read</td>
<td>Update</td>
<td>@DKCUS</td>
</tr>
<tr>
<td>Write to a key label</td>
<td>Read</td>
<td>Control</td>
<td>@DKCUS</td>
</tr>
<tr>
<td>Delete a key label</td>
<td>Read</td>
<td>Control</td>
<td>@DKCUS</td>
</tr>
</tbody>
</table>

Table 3. Granular key label access control
traditionally, bulk encryption on z/OS has come in two basic flavors—high security and high performance:

- For high security, key material is protected by the master keys within the cryptographic coprocessors. This prevents key material from ever appearing in the clear and is referred to as a secure or encrypted key.
- For high performance, a set of instructions called CP Assist for Cryptographic Functions (CPACF) are used to perform cryptographic operations. The clear keys used with CPACF encryption and decryption operations are visible in application storage and within the key store (if they are stored in one).

Conflicting demands between your security policy and business requirements can make the choice between high security and high performance very difficult. But why choose?

**Introducing protected-key CPACF**

Protected-key CPACF is the high-performance and high-security solution for bulk encryption on z/OS. With it, you can now have your cake and eat it too. Protected-key CPACF is able to provide both high performance and high security by taking advantage of the high speed of CPACF while utilizing encrypted keys. It does this by using CPACF wrapping keys to protect the key during CPACF processing instead of passing a clear key. These wrapping keys—one for Advanced Encryption Standard (AES) keys and one for Data Encryption Standard (DES) keys—are analogous to the coprocessor master keys and are visible only to licensed internal code (LIC), never to operating system storage.

Two callable services were enhanced to support protected-key CPACF: Symmetric Key Encipher (CSNBSYE, CSNBSYE1, CSNESYE, CSNESYE1) and Symmetric Key Decipher (CSNBSYD, CSNBSYD1, CSNESYD, CSNESYD1). For many releases, both of these services have accepted labels for the key_identifier parameter when the KEYIDENT keyword is provided in the rule_array. Before protected-key CPACF, this label was restricted to refer to a clear DATA key in the CKDS. With protected-key CPACF enabled, the label may now refer to an encrypted DATA key as well.

So how does it work? Under the covers, ICSF processes a secure key usable by a coprocessor (a CCA encrypted key token) into a secure key usable by CPACF (a CPACF-wrapped key). Each CPACF wrapped key is kept on hand after the first use so it can be used again for a subsequent encryption or decryption request.

To transform a CCA-encrypted key token into a CPACF-wrapped key, ICSF does the following:

1. Determines if the key has already been wrapped for use with CPACF. ICSF maintains a cache of CPACF-wrapped DATA keys. When a label is specified on a call to the Symmetric Key Encipher or Symmetric Key Decipher service, ICSF retrieves the key from the in-storage copy of the CKDS. If it is an encrypted DATA key, ICSF looks for a cached copy and uses it if one is present.

2. Determines if this key is a candidate for wrapping. If the key has not been wrapped for CPACF and cached, ICSF inspects a field in the covering CSFKEYS profile to check for permission. A CSFKEYS profile can contain an ICSF segment, which specifies rules for key use. The SYMCPACFWRAP field of the ICSF segment indicates whether ICSF can rewrap the encrypted key using the CPACF wrapping key. If there is no covering profile, or ICSF(SYMCPACFWRAP(NO)) is set, ICSF does not allow the operation.
3. Requests the wrapping operation. ICSF builds a request to a Crypto Express3 Coprocessor (CEX3C). In the coprocessor, the encrypted DATA key is recovered from under the card master key. The clear form is presented back to the LIC layer, which wraps the clear key value under the corresponding CPACF wrapping key (either AES or DES) before returning the key to operating system storage. At no point during this operation is the clear key value visible in operating system storage.

4. Caches the returned CPACF-wrapped key for future use.

Figure 1 illustrates how ICSF transforms a CCA-encrypted key token into a CPACF-wrapped key. Caching the CPACF-wrapped key value is crucial to the performance of protected-key CPACF operations. ICSF only creates the CPACF-wrapped key value when a key is initially requested by label. When it is created that wrapped key value is then cached. Because performance in CPACF is orders of magnitude faster than operations on a coprocessor, maintaining a cache of CPACF-wrapped key values removes the overhead of having to recreate that wrapped form for future requests.

Protected-key CPACF has the following requirements:

- z10™ EC or z10™ BC Server with CPACF code enabled on the LPAR (feature code 3863)
- CEX3C with the November 2009 or later LIC
- z/OS V1R10 or z/OS V1R11 with Cryptographic Support for z/OS V1R9-R11 (ICSF WD#9 - HCR7770) and new function SPEs (RACF APAR OA29193 and SAF APAR OA29194).

For more information

The Symmetric Key Encipher and Symmetric Key Decipher callable services are described in z/OS Cryptographic Services ICSF Application Programmer’s Guide, SA22-7522-13.

For more information on the SYMCAPACFWRAP field of the ICSF segment of CSFKEYS profiles, see z/OS Cryptographic Services ICSF Administrator’s Guide, SA22-7521-14.
Share and share alike

How ICSF keys are shared in a sysplex

BY RONALD EDICK

When systems are operated in a sysplex, it is important to maintain the consistency of the key stores across the systems in the sysplex. Parallel Sysplex® harnesses multiple z/OS systems into a single computing facility. The sysplex permits the sharing of resources across the systems. The Integrated Cryptographic Service Facility (ICSF) takes advantage of the sysplex facilities to share its cryptographic key data.

ICSF requires cryptographic keys for its security functions. The cryptographic keys are retained in key store datasets. These key stores are dynamic. Keys can be added, deleted, or updated during ICSF operation.

ICSF key data architecture

ICSF maintains three key data sets (KDSs). The cryptographic key data set (CKDS) holds symmetric (DES or AES) key tokens, the public key data set (PKDS) holds asymmetric (PKA) key tokens, and the token data set (TKDS) holds PKCS11 objects. The contents of these data sets are loaded into the ICSF address space during activation of ICSF. The CKDS is read into the CKT, the PKDS is read into the PKT, and the TKDS is read into the TKT. Each z/OS system that starts ICSF loads its own copy of the KDS as illustrated in Figure 1.

ICSF performs most of its cryptographic operations by accessing the in-storage copy of the key data (CKT, PKT, or TKT), not the data set (CKDS, PKDS or TKDS). A key update to the data set by one system is not communicated to other systems in the sysplex unless the information is explicitly shared across the sysplex.

Defining key stores and key store sharing

ICSF provides an OPTIONS data set for configuring ICSF. The options include the definition of the key stores, and how those key stores are to be shared in a sysplex. The definitions are independent. Any or all of the data sets can be shared.

The definitions are illustrated for the CKDS in Figure 2, but the PKDS and TKDS definitions are similar. The CKDS can be specified by using the CKDSN option. It can be shared across the sysplex by adding the SYSPLEXCKDS option parameter.
Sysplex sharing

ICSF uses the cross-system Coupling Facility (XCF) to share key store data. XCF provides the capabilities for the members of a sysplex to communicate with each other and exchange information. Groups are defined by XCF among which information can be shared. ICSF can maintain a group for the CKDS, a group for the PKDS, and a group for the TKDS. Each group contains the KDS name as part of the XCF group identifier (resource name).

The CKDS is used to illustrate this concept. Assume that there is a sysplex with four members using ICSF. Assume ICSF on members 1 and 2 use CKDS A and that ICSF on members 3 and 4 use CKDS B. This sysplex has two ICSF sysplex groups. Members 1 and 2 form group A and members 3 and 4 form group B. Updates to CKDS A are shared by members 1 and 2. Updates to CKDS B are shared by members 3 and 4. Members 1 and 2 are unaware of updates to CKDS B. Members 3 and 4 are unaware of updates to CKDS A. The CKDS is used to illustrate this concept, as shown in Figure 3.

Sysplex rearrangement

Sysplex groups may be rearranged dynamically. The groups can be changed by changing the KDS in use. The KDS can be changed by either refreshing the KDS or restarting ICSF.

The CKDS can be used to illustrate this concept. Assume our original configuration of four systems arranged in two sysplex groups. At some point, system 3 does a CKDS refresh and changes from CKDS B to CKDS A. The sysplex groups are now rearranged with group A consisting of systems 1, 2, and 3. Group B now consists only of system 4. The CKDS is used to illustrate this concept, as shown in Figure 4.
Sysplex communication
The Cross System Coupling Facility (XCF) is a component of z/OS that facilitates communication between systems. XCF messaging commands are used to send and receive messages across the sysplex. The messaging allows ICSF to manage the system-wide enqueues necessary to serialize the KDS update process and to transmit the KDS updates across the sysplex.

Update mechanics
Updating the KDSs in a sysplex requires serialization and handshakes between sysplex members. The mechanics of the process involve enqueues, latches, and XCF messaging. The process is illustrated for the CKDS but all KDSs follow the same procedure.

When a CKDS update is initiated, an exclusive sysplex-wide I/O enqueue is acquired. The exclusive enqueue prohibits any other system from accessing the data set while the originator operates on it. The enqueue is held for a resource with a queue name (QNAME) of SYSZCKT and a resource name (RNAME) of the CKDS data set name. Once a system has established the I/O enqueue, the update process begins. The update process consists of preparing the sysplex for the update, updating the CKDS, and notifying the member systems of the change in key data.

Preparation phase
The first step in the update process is the preparation phase. During the preparation phase, the members of the sysplex are notified that an update is to occur and are asked to acknowledge they are ready. The originator requests exclusive use of a preparation enqueue. This enqueue is normally shared by the sysplex members. The originator broadcasts a message to all member of the group. The members respond by temporarily releasing their hold on the preparation enqueue. When the originator’s request for exclusive use of the enqueue is granted, it knows that all members are ready for the update. The preparation resource for the CKDS is SYSZCKDS/ckdsname.

CKDS update
The originator updates the CKDS. It then proceeds to the completion phase to notify the sysplex members of the change in key data.

Completion phase
Before the release of the preparation enqueue, the sysplex members obtain a completion enqueue which is held shared until the update is completed on that member. For the CKDS, this enqueue resource is SYSZCKUP/ckdsname.

The completion phase is entered when the originator broadcasts a second XCF message containing the update of the record. The sysplex members receive the message and update their in-storage copy of the KDS. When a member completes its update, it releases its hold on the completion enqueue (SYSZCKUP/ckdsname).

Update done
The KDS update is done when all systems in the sysplex have updated their in-storage copies of the KDS. The originator knows that this has occurred when it is able to obtain an exclusive enqueue on the completion resource. When that has occurred, the originator releases the I/O enqueue and completion enqueue and returns to the caller. The key store update is finished at this point.

Conclusion
The user has the option of sharing any or all of their ICSF KDSs in a sysplex. If a KDS is not shared, the system operates in isolation and requires that the in-storage copy of the KDS be reloaded (refreshed) to obtain updates to the KDS. If the KDS is shared, any update to the KDS anywhere in the sysplex will be reflected on all members that share it.

PTF UK52956 reduces the number of steps needed to configure an instance of IBM z/OS Management Facility (z/OSMF) on your system. This PTF also introduces new options to help simplify the configuration process. The new -overridefile option, for example, allows you to use an override file to update variables in the configuration file with your installation-specific values. Also, the new -fastpath option allows you to run the configuration script “quietly,” rather than interactively through a series of script prompts.

For details on these enhancements, see the latest edition of z/OSMF User’s Guide, SA38-0652-02, which is available at the following Web site:

ibm.com/systems/z/os/zos/zosmf/

Installations not currently at the PTF UK52956 level should apply this PTF before configuring z/OSMF. If your installation has already configured an instance of z/OSMF, install this PTF prior to any subsequent configurations that you perform.
Extracting RACF data from REXX™ programs has never been easy. In fact, it’s been like pulling teeth. Without anesthesia. Prior to z/OS V1R11, there have been two ways to retrieve RACF data using REXX. You could:

- Capture the output of a RACF TSO command (such as LISTUSER or LISTGRP) using the OUTTRAP function, and parse out the information you needed. Unfortunately, because a TSO command is not an official programming interface, the output from these commands could be ambiguous and subject to change. This could make it hard for the OUTTRAP function to parse.

- Code an Assembler program that used RACROUTE REQUEST=EXTRACT macros to get the data, then converted that data from its internal format on the database, and finally used a TSO/E service to write the values into REXX variables. This second option was rather complex, so most people took their chances parsing RACF command output using the OUTTRAP function.

Novocaine and nitrous oxide

Over time, both approaches to extracting RACF data using REXX were improved. For those coding Assembler programs, RACF introduced user and group extract functions in the R_admin callable service (IRRSEQ00). This provided important improvements over RACROUTE REQUEST=EXTRACT. The group extract functions could be called from problem state programs, the output format was now a supported programming interface, and the output was in character format. This second option was rather complex, so most people took their chances parsing RACF command output using the OUTTRAP function.

With IRRXUTIL, extracting RACF data is no longer like pulling teeth

With z/OS V1R11, a true REXX interface to the R_admin callable service is available as a fully supported function in RACF. The new program is called IRRXUTIL. It reads profile information using the R_admin callable service, and creates a whole bunch of REXX stem variables to contain the profile data (including custom fields!). What’s more, IRRXUTIL is able to return general resource and SETROPTS data in addition to user and group data. Now when your REXX program extracts RACF data, it’s no longer like pulling teeth.

Because IRRXUTIL uses the R_admin callable service in problem state to retrieve data from the RACF database, callers must be authorized to use the extract functions of the R_admin callable service. The security administrator uses profiles in the FACILITY class to control who has access. Even when the user has access to the R_admin callable service, it still returns only the profiles and fields that the user is allowed to view. The access rules are the same as those enforced by the RACF LIST commands (LISTUSER, LISTGRP, RLST, SETROPTS LIST).

Figure 1 shows a simple REXX program that extracts RACF profile data using the IRRXUTIL program. Line 2 of this example calls IRRXUTIL to extract profile data for user “U1” and to place that data in REXX stem variable "RACF". Notice how the program checks the return code on line 2. IRRXUTIL returns a series of values as a return code. If the first value is 0, the service was successful.

```
/* REXX */
1. myrc=IRRXUTIL("EXTRACT","USER","U1","RACF",""")
2. if (word(myrc,1)<>") then do
   3.     say "Error: "myrc
   4.     exit
   5. end
6. say "User ID is "RACF.profile
7. say "Owner is "RACF.BASE.OWNER.1
8. say "UID is "RACF.OMVS.UID.1
9. say "Default group is "RACF.BASE.DFLTGRP.1
10. do i=1 to RACF.BASE.CGROUP.0
11.   say " Connect Group "i" "RACF.BASE.CGROUP.i
12. end
```

Figure 1. REXX program using IRRXUTIL to extract RACF profile data

From here, the REXX program simply accesses the data it needs, referring to it by segment and field name, in the newly-created “RACF” stem variable. The example accesses both standard (single value) fields on lines 7, 8, and 9, and repeating fields on
lines 10 and 11. Note that for any field, the variable with the suffix ".0" contains the number of values in the field, and the suffix ".n" contains the nth value of the field.

In addition to field values, IRRXUTIL also sets a number of variables in the stem containing metadata about the data returned. For example, fields can be identified as being BOOLEAN or OUTPUTONLY. The metadata could also specify other characteristics, such as the data being REPEATING data. This metadata is useful for applications that need to process data in a profile without knowing exactly what data is returned in advance. For example, an application may want to export the data into another format, or display it in a report.

Figure 2 shows a REXX program that dumps all data found in the profile, without knowing anything about which segments and fields exist in the profile.

```
/* REXX */
myrc=IRRXUTIL("EXTRACT","GROUP","G1","RACF",""") /* call IRRXUTIL */
say "ID is "RACF.profile /* Display profile name */
do i=1 to RACF.0 /* Iterate through segments */
   say segname "="RACF.i /* set segment name */
   do j=1 to RACF.segname.0 /* iterate fields in segment */
      say fieldname=" "RACF.segname.j /* set fieldname */
      do k=1 to RACF.segname.fieldname.0 /* iterate values in field */
         say segname "="fieldName "="RACF.segname.fieldname.k /* print */
      end
   end /* values */
end /* fields */
end /* segments */
```

Figure 2. program using IRRXUTIL to extract all the data in a profile

Extracting the next profile

The IRRXUTIL program provides two functions. So far, we've shown you the EXTRACT function, which retrieves a specified profile from the RACF database. In addition to this function, a program can use the extract next function (EXTRACTN) to extract the profile that alphabetically follows the specified profile. For example, an attempt to EXTRACTN profile G1 may return data for profile G2 (assuming a profile G2 is in the RACF database). Specifying a profile name of " " (space) will return the very first profile in the class. The extract next function can be very powerful when used iteratively (but see the friendly warning later in this article).

Some things you can easily do

Now that the IRRXUTIL program has made it so much easier for REXX programs to extract RACF data, let's consider some things you can do.

Roll your own RACF list command

Have you ever wanted RACF command output to be formatted differently? Consider using IRRXUTIL to display just the information you want in the format you desire. For example, you can alphabetize the user's connect groups in LISTUSER. Or you can display a subset of SETROPTS information so you don't have to wade through three pages of irrelevant command output. In fact, we've done the heavy lifting for some of these just to get you started. For samples, see the link for the RACF Downloads Web site at the end of this article.

Populate ISPF panels and tables

You can use the IRRXUTIL program to sort RACF profiles and display the profile information in the format of your choosing. In fact, the RACF ISPF panels for custom fields and the ICSF segment of the general resource profile use the IRRXUTIL program to prime the panels with existing values.

New password and phrase exit

There is a sample REXX-based new password exit (on the RACF Downloads Web site) and a new password phrase exit (shipped in SAMPLIB). Some customers have expressed a desire to apply more stringent password rules to privileged users. Your REXX exit could, for example, extract the SPECIAL attribute and enforce a longer minimum password length if the value is TRUE.

Other things you can easily do (but really shouldn't)

The IRRXUTIL program is powerful, but with power comes responsibility. Here are some friendly warnings.

Just because you can read an entire class, or the entire RACF database, into REXX variables doesn't mean you should

What you do with your memory is your own business, but keep in mind that the R_admin callable service (and by extension the IRRXUTIL program) drives I/O against the RACF database. That I/O will be competing against all the other RACF database I/O. RACF's true mission in life is to authenticate users and perform access decisions, and to do so very rapidly. Don't get in its way.

You may be tempted to make your own security decisions. Don't do this!

You may think that only access lists and universal access are involved in a security decision, and with a few lines of REXX code, you can inspect the contents of a general resource profile and decide for yourself whether a user has access. But there's much more to it than that. (Check out Appendix E of z/OS Security Server RACF Security Administrator's Guide, SA22-7683, and its 31 authorization steps to get a feel for what we are saying.) And then there's the audit trail that RACF is expected to leave when authorization decisions are made. If you really want an application that operates in the true spirit of z/OS security, define profiles in RACF and use RACROUTE REQUEST=AUTH to drive access requests against them. IRRXUTIL should not be used for this.

For more information!

IRRXUTIL is described in detail in z/OS Security Server RACF Macros and Interfaces, SA22-7682. Refer to this documentation for complete information about the service, including all inputs and outputs. This document also provides diagnosis information, SETROPTS considerations, repeating field details, and information on reading multiple profiles into a REXX stem.

For sample code, see the RACF Downloads Web site:

There was a time when programmers used pocket protectors filled to capacity with pens and pencils of every color. Their media was punch cards. They knew how to use slide rules. And they wrote programs in assembler. Their legacy is the multiple volumes of assembler macros, many of which we still enjoy and use to this very day.

With the natural progression to object oriented languages, many programmers thought they would never have to deal with assembler ever again. But sometimes, the only way to get the job done right is to take a journey down into the primordial ooze of z/OS. But how do you get from Java to an assembler interface? Fear not! Getting there and back is the easy part. Now, how are your assembler coding skills?

We’ll begin our journey in Java, and extend through the Java Native Interface (JNI) to the C source that makes the call to assembler and returns the results. Let’s pretend that we want an assembler routine to add two numbers. We’ll take the result from the assembler routine, and either store it in a buffer or return it through linkage conventions. This is a simple example, but it shows the process of designing and coding a Java program that accesses assembler.

1. We begin with a simple Java method called addnumbers.java that will manage the trip through C and assembler and back again. Because this method calls an assembler routine that you own, it requires the Java Virtual Machine (JVM) to load your dynamic link library (DLL). The DLL contains the compiled C and assembler source linked together. The Java class addnumbers does not reside inside the DLL:

```java
/* addnumbers.java - This Java example uses a JNI interface that resolves to a c module calling an assembler module. */

public class addnumbers
{
    /* JVM loads your dll */
    static {
        System.loadLibrary("addnum");
    }

    public static void main(String[] args) {
        if (args.length != 2) {
            System.out.println("Usage:");
            System.out.println("\tjava addnumbers <number1 number2>");
            return;
        }
        int n1 = Integer.parseInt(args[0]);
        int n2 = Integer.parseInt(args[1]);
        int n3 = 0;
        addnumbers t = new addnumbers();
        System.out.println("addnumber.java: Call JNI routine to add <" + n1 + "> and <" + n2 + ">");
        n3 = t.doAdd(n1, n2);
        System.out.println("addnumber.java: results <" + n3 + ">");
    }

    public native int doAdd(int a, int b);
}
```

/* addnumbers.java - This Java example uses a JNI interface that resolves to a c module calling an assembler module. */

public class addnumbers
{
    /* JVM loads your dll */
    static {
        System.loadLibrary("addnum");
    }

    public static void main(String[] args) {
        if (args.length != 2) {
            System.out.println("Usage:");
            System.out.println("\tjava addnumbers <number1 number2>");
            return;
        }
        int n1 = Integer.parseInt(args[0]);
        int n2 = Integer.parseInt(args[1]);
        int n3 = 0;
        addnumbers t = new addnumbers();
        System.out.println("addnumber.java: Call JNI routine to add <" + n1 + "> and <" + n2 + ">");
        n3 = t.doAdd(n1, n2);
        System.out.println("addnumber.java: results <" + n3 + ">");
    }

    public native int doAdd(int a, int b);
}
2. Next we have the C code that takes the input from the Java method and passes the two numbers to the assembler routine. It also passes the assembler routine a buffer address to store the output in:

```c
/* addnumbers.c - An example of using JNI to get to an assembler * routine. *
 * * What the c routine does. *
 * 1. Call assembler routine that takes three input parameters. The *
 * assembler routine adds parms 1 and 2 and places result in *
 * parm 3 and also return results in register 3. *
 * 2. Return result to addnumbers.java */

#define _POSIX_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include "addnumbers.h" /* Native method declarations. Generated */
/* by javah -jni addnumbers */

/* Define entry to assembler routine and map */
/* it to a more meaningful name. */
#pragma linkage (ADDEM, OS)
#pragma map(add_numbers,"ADDEM")
int add_numbers (int n1, int n2, void* buffer);

JNIFLAGOJ ICALL Java_addnumbers_doAdd(JNIEnv *env, jobject obj,
    jint n1, jint n2)
{
    int result = 0;
    int buff = 0;

    printf("\naddnumbers.c: Received ints <%d> and <%d>\n", n1, n2);
    printf("\naddnumbers.c: Address of our buffer <x%8.8X>\n", &buff);

    /* Make call to assembler routine *
    * Pass in the two numbers to add and address of a buffer *
    * to also store results. */
    result = add_numbers(n1, n2, (void *)&buff);

    printf("\naddnumbers.c: Result <%d> returned via XPLINK conventions\n", result);
    printf("\naddnumbers.c: Buffer <%d> returned via storage update\n", buff);
    printf("\naddnumbers.c: Returning result <%d>\n\n", result);
    return result;
}
```

When you run the build.sh shell script in step 5 below, the javah command in addnumbers.java creates the following JNI header file required by the C source:

```c
/#* DO NOT EDIT THIS FILE - it is machine
generated */
#include <jni.h>
/* Header for class addnumbers */

#endif

#define Included_addnumbers
#define _Included_addnumbers
#endif
extern "C" {
#endif
/* *
 * Class: addnumbers *
 * Method: doAdd *
 * Signature: (II)I *
 */
JNIFLAGOJ ICALL Java_addnumbers_doAdd
    (JNIEnv *, jobject, jint, jint);
#endif
#endif
```
3. Next, we have the assembler routine that adds our two numbers and places the result into the specified buffer address. This routine also uses the XPLINK convention to return the result in register 3. This example shows a 64-bit routine—the code uses AIF statements that test for the compile time flag B64:

```
TITLE 'ADDEM – Add a+b in 31 or 64 bit mode'
AIF ("&SYSPARM' EQ 'B64'").ENTRY64
.ENTRY31 ANOP
ADDEM EDCXPRLG BASEREG=R6 entry AMODE=31
  * Add numbers in AMODE=31
  AR R1,R2     add regs 1 and 2
  ST R1,0,(R3) place results into buffer
  LR R3,R1     place results into return reg
  AGO .EXIT31
.ENTRY64 ANOP
ADDEM CELQFPLG BASEREG=R6 entry AMODE=64
  * Add numbers in AMODE=64
  AGR R1,R2     add regs 1 and 2
  STG R1,0,(R3) place results into buffer
  LRG R3,R1     place results into return reg
.EXIT31 ANOP
AIF ("&SYSPARM' EQ 'B64'").EXIT64
EDCXEPLG     return AMODE=31
AGO .EXIT
.EXIT64 ANOP
CELQEPLG     return AMODE=64
.EXIT ANOP
* Symbols and Reg Equates
R1       EQU   1
R2       EQU   2
R3       EQU   3
R6       EQU   6
END
```

4. Now it's time to fine tune the build.sh shell script you'll use to run addnumbers.java. The build.sh shell script creates the addnumbers.class, the JNI header, and DLL (libaddnum.so) Make sure that the environment variable, JAVA_HOME, points to your locally installed JDK:

```
#!/bin/ksh
java -version
# compile java
javac addnumbers.java
# create JNI header
javah -jni addnumbers
# compile assembler source.
# -W "a,SYSPARM(B64)"
  c89 -W "a,offset,list,goff"
    -o ADDEM.o ADDEM.s > ADDEM.list
# compile c source
# -W "c,lp64,warn64"
  c89 -c
    -W0,LIST,SOURCE
    -W "c,dll,xplink,exportall,langlvl(extended),float(ieee)"
    -I${JAVA_HOME}/include -I. 
    -o addnumbers.o addnumbers.c > addnumbers.list
# link dll
# -W "l,lp64,AMODE=64"
  c89 -o libaddnum.so 
    -W l,dll,xplink 
      addnumbers.o ADDEM.o 
      ${JAVA_HOME}/bin/j9vm/libjvm.x
> linkage.list
```
To create a 64-bit version of the DLL in the build.sh shell script, copy and un-comment the supplied 64-bit compiler options for each c89 command, as shown in the following example:

```
#   -W "a,SYSPARM(B64)" \
c89 -W "a,offset,list,goff" \
   -o ADDEM.o ADDEM.s > ADDEM.list
```

5. You’re ready to run the build.sh shell script:

```
> build.sh
```

6. build.sh builds the addnumbers class. Now we need a shell script to run in the class. Here’s our runsample.sh shell script:

```
#!/bin/ksh
if test x$JAVA_HOME = x
then
   echo \"\nPlease set JAVA_HOME to the location of your JDK.\"
   echo \"\nFor example: \"
   echo \"\nexport JAVA_HOME=/usr/lpp/java/IBM/J5.0\"
   exit 1
fi

# Update the path envars. Assumes execution from the same
# directory the sample was built in.
export LIBPATH=$LIBPATH:.
export CLASSPATH=$CLASSPATH:.
java addnumbers 1234 5678
```

Finally, issue runsample.sh as follows to run the addnumbers sample:

```
runsample.sh
```

When you run runsample.sh, you’ll see the progression from the addnumbers.java to the addnumbers.c source, and the results from the assembler ADDEM.s routine:

```
addnumber.java: Calling JNI routine to add <1234> and <5678>
addnumbers.c: Received ints <1234> and <5678>
addnumbers.c: Address of our buffer <x0C7D9284>
addnumbers.c: Result <6912> returned via XPLINK conventions
addnumbers.c: Buffer <6912> returned via storage update
addnumbers.c: Returning result <6912>
addnumber.java: Results <6912>
```

Now that you’ve got the idea, can use these samples to call lots of assembler routines from Java code.

**Design considerations**

Consider the following when designing a Java application to call an assembler routine:

- Look carefully at the linkage convention used in the assembler routine you want to use. If it doesn’t use XPLINK, you must code the C source pragma statements differently.
- If an assembler routine requires a work area or special storage you can set up the storage from the C source and then pass it to the assembler routine. You can do the same thing for lengthy parameter lists. (Note however, that XPLINK works best for three parameters or less!) This can help keep your assembler coding to a minimum.
- Before you start coding, think about what your program requires when the assembler routine returns control. If it is simply a single result, you can use linkage conventions to place the result in the proper return register. Or, as our example shows, you can place the results in a specified buffer. Use the compile flag SYSPARM to structure your assembler code as either 31- or 64-bit. A 31- or 64-bit JVM fails when it tries to load a DLL with an unmatching AMODE version.
- Java is not APF-authorized. A routine fails if it requires APF authorization, tries to change keys, or changes to supervisor state.

So now you know how to write a Java application that calls an assembler routine. Perhaps it’s time to shop for your very own pocket protector!
This is another article in an ongoing series that answers common catalog-related questions. I have worked with catalogs for over 25 years as a developer, designer, and architect. I hope you find these questions and answers beneficial.

If you have a question, please send it to: branches@us.ibm.com

This is from a reader of z/OS Hot Topics Issue 21, (GA22-7501-17):

(Q) At last, some catalog news and updates in the z/OS Hot Topics Newsletter. I must say that I really enjoyed your topic, and I am really looking forward to the next one! I have a question: How do you determine the size of a catalog without a tool such as Tivoli Advanced Catalog Management (which we use). And what determines the GB limit? Is it the amount of entries in the catalog? Can you elaborate on this topic as I am genuinely interested in understanding this?

(A) Catalog size is strictly a matter of the number of entries in it. You cannot calculate the exact size of a catalog because the formula is dependent on a number of things, and not all entries in a catalog are in fixed length. For estimation, see the topic “Estimating Space Requirements for the BCS” in DFSMS: Managing Catalogs, SC26-7409. It contains a table that provides the estimated length of each entry type. Use the table and a count of each type of entry in the output of a LISTCAT ALL command of the catalog to calculate how much space is necessary. This space calculation is the minimum requirement. The book topic also describes how to estimate the free space and secondary amounts.

Part of the reason for the 4-GB limit is that VSAM uses a 4-byte relative byte address (RBA) to find records. 4 GB is the largest number that can be contained in 4 bytes. To go beyond the 4-GB limit, VSAM extended addressable data sets use relative control interval numbers (RCIs) to find records. Thus, the size of the largest VSAM data set that uses RCIs is 4 GB times the data set control interval (CI) size, very large, indeed, if you are using 32-KB CI sizes.

Another reader writes:

(Q) I am attempting to alter the buffer number (BUFNI) for my catalog index. When I issue ALTER ICFUCAT BUFNI(6), I get IDC3190I 'BUFNI' PARAMETER INVALID WITH ENTRY TYPE. How do I ALTER the catalog index buffer number?

(A) First, the ALTER fails because IDCAMS is trying to update the user catalog connector record in the master catalog. User catalog connector records do not have BUFNI values. You need to alter the BUFNI for the user catalog itself.

You can simply add a catalog parameter to the ALTER to specify the catalog that you want to change. IDCAMS attempts to change the DATA component, and the request will fail because the BUFNI parameter applies only to the INDEX component. To change any index attribute, you must specify the catalog index name, which is the same as the catalog data component name appended with “CATINDEX.”

To alter the catalog index information, be sure to specify the ALTER command as follows:

```
ALTER ICFUCAT.CATINDEX BUFNI(6) CAT(ICFUCAT)
```

where ICFUCAT is the user catalog name and ICFUCAT.CATINDEX is the catalog index name.

(Q) Is it at all possible to uncatalog a base VSAM cluster without uncataloging all of the associated alternate indexes (AIXs)? Do I require a special override or such?

(A) By “uncatalog,” I assume you mean DELETE NOSCRATCH. The answer is “No.” The AIX information is in a sub-record within the base VSAM cluster record. When you delete the base VSAM cluster record, you also delete all the associated records, including AIX and PATH records. There is no way to override this deletion. When the base VSAM cluster record and the associated sub-records are removed with DELETE NOSCRATCH, the physical data sets still exist on the volumes and can be recataloged using DEFINE RECATALOG. You will have to redefine any PATHs with the DEFINE PATH command.
(Q) Can I compress a VSAM entry-sequenced data set (ESDS)?

(A) No, a VSAM ESDS is not compressible because a record size change during update cannot be tolerated by VSAM ESDS processing. VSAM allows you to update ESDS records in place if the record size does not change. It is highly unlikely that a record is compressed to the same size when its contents are changed.

If you think a read-only compressed ESDS would be especially valuable to your business, I encourage you to submit a requirement for this function.

(Q) There was a change to VSAM space allocation during the z/OS V1R10 timeframe. I used to allocate a two track data set, which was sufficient to hold my data. Now, in z/OS V1R10, the same DEFINE gets three tracks. Can you explain to me the why the change, and if this is correct?

(A) In z/OS V1R10, there was a change to VSAM space allocation for extended addressable volume (EAV) support. For EAV cylinder-managed space (CMS), DASD space is obtained in 21-cylinder (or 315-track) chunks (called multi-cylinder units). In anticipation that a VSAM data set might extend into the CMS, and because the control area size must be a multiple of the allocation amount, you round the allocation size up to an even multiple of the CMS chunk size. The rounding only occurs for track allocations (cylinder allocations are already an even multiple). Because 3, 5, 7, 9, and 15 are all multiples of 315, 2 tracks round to 3, 4 tracks round to 5, 6 tracks round to 7, 8 tracks round to 9, and 10 through 14 tracks round to 15.

CSSMTP does not provide a full replacement for SMTPD, but supports most features that allow you to move your current mail jobs seamlessly. For example, CSSMTP does not provide SMTP server support to allow you to receive mail on z/OS, but in most cases, z/OS mail requires SMTP client support rather than server support. The primary role for CSSMTP is to perform as a client-side network job entry (NJE) mail gateway for forwarding mail from z/OS.

Do yourself a favor and find out more about what CSSMTP provides by reading the following topics in z/OS V1R11 Communications Server IP Configuration Guide, SC31-8775-15:

- “Configuring the CSSMTP application”
- “Differences between CSSMTP and SMTPD”

If you are using the SMTPD mail application to send mail from the JES spool to Internet destinations, you must try out the new Communication Server Simple Mail Transfer Protocol (CSSMTP) mail application provided with z/OS V1R11. Here are a few of the benefits:

- Easy to configure and start—you can enter the S CSSMTP command on an MVS™ console or from System Display and Search Facility (SDSF)
- Quick way to send your mail—thanks to multi-threaded design, which removes the bottlenecks you might have experienced with SMTPD
- No need to reserve DASD—CSSMTP processes mail directly from the JES spool data set and sends the mail to the target server
- Useful reporting features—generates a report for the mail administrator and lets them decide what actions to take for errors.
It’s the zFavorites for System z credit card CD! You’re gonna love this! It has all sorts of helpful Web links, like those for:

- Hardcopy
- Operating systems
- Software
- Language and tools
- ISV development and applications
- Product documentation
- Marketing information
- Education
- Support
- Links to FREE downloads
- IBM Redbooks® sampler
- WebSphere Application Server
- XML.

To use the CD insert it in any standard CD device, and it should start automatically. If it does not, click Start > Run, and then type x:\index.htm (where x is your CD drive letter) and press Enter.

Additional copies of zFavorites CD (GK3T-4331-16) are separately orderable.
Managing your I/O configuration takes time and skill. A good I/O configuration will factor in both performance as well as availability needs. For example, the configuration should assure that no single point of failure (SPOF) exists in the paths between system images and storage devices. If all paths to a device go through the same FICON® director (switch) or the same host adapter on a storage subsystem, the failure of that component of the I/O topology could result in a system outage.

The I/O configuration should also accommodate the loss of components while still maintaining adequate performance. So business critical workloads might require four or more paths, where none of those paths share an SPOF, in order to assure that no more than 25% of the channel bandwidth is lost because of any single failure.

The number of channels that are defined to a set of devices must also be carefully considered. Sufficient channels should be available for peak activity that might occur periodically, such as special end-of-month, or end-of-quarter processing. This process is often iterative. Metrics are collected and analyzed, and an I/O configuration is defined. Over time, as workloads evolve, the configuration might not be optimal anymore, and the process must be repeated. The addition of new FICON directors, servers, or storage subsystems also means that the I/O configuration must be re-examined in order to make best use of all resources.

Even when the I/O is configured for peak activity, hardware failures and unexpected peaks can result in unacceptable I/O response time.

Wouldn’t it be nice if z/OS could automatically react to changes in your I/O configuration, such as unanticipated workload spikes and failures in the I/O fabric resulting in loss of paths?

Pass the remote!

Well, you’re in luck! In z/OS V1R11, Dynamic Channel-Path Management (DCM) does this by letting customers identify channels that they wish to be “managed.” Managed channels are channels that you have not assigned to any specific controller, but instead are viewed as a pool of channels (or CHPIDs) that can be assigned dynamically to control units at the discretion of the system.

When you assign channels to the pool of managed channels, the system can assign them to respond to peaks in control unit demand for I/O channel bandwidth.

Allowing the system to manage channels helps to simplify I/O configuration planning and definition and reduce the time it takes to manage z/OS I/O configuration performance. It is designed to balance I/O channel resources dynamically in response to changing workload demands, make more efficient use of channel resources, and help you enhance availability.

DCM concepts

To understand DCM, you should become familiar with the following concepts:

- A non-managed channel, also known as a static channel, is one that is assigned to one or more control units in the I/O definition file (IODF). DCM cannot reassign non-managed channels. A non-managed channel can be shared or spanned across logical channel subsystems (LCSSs), and can be connected to the control units directly (point-to-point) or by using a switch (switched point-to-point).

- A managed channel is one that is dynamically assigned by DCM to a control unit based on workload and availability needs. Unlike non-managed channels, a managed channel is not assigned to a specific set of control units in the IODF. Managed channels can only be shared, not spanned, and they must be connected to a switch. See Modifying the I/O configuration using HCD for more information.

- A managed control unit is one that allows managed channels to be assigned by DCM. Managed control units are defined by specifying one or more asterisks (“*”) in the list of channel path IDs for the control unit. See Modifying the I/O configuration using HCD for more information.

- A non-managed control unit is one that is defined with only non-managed channels.

- An I/O cluster consists of all z/OS images (logical partitions) that are members of the same sysplex and reside in the same central electronic complex (CEC) and LCSS. A managed channel is owned by an I/O cluster; that is, it can only be assigned to z/OS images within the I/O cluster.

- A control unit group consists of a set of control units that have the exact same set of non-managed channels defined. When DCM assigns a managed channel, it assigns it to all control units in the group.
Configuring FICON DCM

Before you can use DCM, you need to modify your I/O configuration to define managed channels and control units and to define the switches connected to the managed channels. Here is how to modify your I/O configuration using Hardware Configuration Definition (HCD).

Modifying the I/O configuration using HCD

To define a managed channel, you can either add new channels to the I/O configuration or convert existing non-managed channels to managed channels. If you are converting an existing channel, it must be removed from all control unit definitions first. A managed channel cannot be defined in the channel path list for any existing control units.

Figure 1 shows an example of adding a new managed channel. Ensure that the managed field is YES, the operation mode is SHR, and the I/O cluster name is the name of the sysplex that will own the managed channel. Because managed channels must be switch-connected, the switch ID must be specified in the “Dynamic entry switch ID” field. It’s good practice to specify the “Entry switch ID” and “Entry port” fields as well.

To define a managed control unit, you can either add a new control unit definition or convert an existing non-managed control unit to be managed. Only DASD control units are supported by DCM. This restriction is not enforced by HCD. If a non-DASD control unit is defined, it will not be managed by DCM.

Figure 2 shows an example of adding a new managed control unit. A managed control unit requires a minimum of one non-managed channel to be defined to allow DCM to identify it as a control unit that it needs to manage. However, define at least two non-managed channels for availability reasons.

To allow DCM to add managed channels to the control unit, specify an asterisk in the list of channel path IDs, instead of a CHPID, and leave the “Link address” field blank. The number of asterisks defines the maximum number of managed CHPIDs that may be added by DCM. We recommend specifying asterisks for the remaining channel paths to allow DCM to add as many channels as it needs.

In order for DCM to obtain topology information, consider the following:
- The switches connected to the managed channels must be defined as devices in the I/O configuration and varied online to z/OS.
- The Control Unit Port (CUP) feature must also be installed on each switch.

Once the changes have been made, activate the new I/O configuration, configure the managed channels online, and vary the switch devices online. z/OS enables DCM automatically when it detects that there are managed channels and control units. No parmlib changes or commands are required to enable DCM.

Wouldn’t it be nice if z/OS could automatically react to changes in your I/O configuration?
How FICON DCM works
During IPL, DCM builds tables that represent the topology of the system. This information is collected from the channel subsystem, the switches, and the control units. DCM uses this information to determine what changes can be made and to determine their availability characteristics.

Periodically, DCM collects performance information for all systems in the I/O cluster. This information is used by DCM to determine when a change must be made and whether a particular change will yield the correct results. DCM also monitors the I/O configuration for hot spots (that is, high channel utilization) and availability need conditions (that is, less than two online paths). When a hot spot or availability need condition is detected, DCM uses the topology information to create a list of changes that can be made to correct the problem. It then selects the best change to make based on its performance and availability characteristics, dynamically changes the I/O configuration, and varies the new path online to all of the affected devices.

Requirements for FICON DCM
FICON DCM requires that z/OS V1R11 with the PTF for APAR OA30485 be installed on all systems in the I/O cluster where you want to run DCM. All System z processors and FICON channels are supported. FICON DCM requires a 2105 model 800 storage control unit or higher. All switches are supported; however, the switches connected to the managed channels must have the control unit port (CUP) feature installed.

Further information
For more information about FICON DCM, see IBM techdoc FICON DCM for System Programmers, which can be found at the following Web site:

ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP101544

You can control the way HCD (Hardware Configuration Definition) performs I/O configuration tasks by setting options in your HCD profile. Instead of using the default options, you can tailor the options to match your individual preferences. For example, you can set your own options in your HCD profile data set to do the following tasks:

• Define new devices
• Customize IOCP deck migration
• Display the colors you prefer in the graphical interface.

In z/OS V1R11, HCD supports a new selection in the HCD dialog, called the Profile Options dialog, to let you change the options in your HCD profile. For example, on this panel you can:

• Display existing values
• Check the changes for validity
• Find help information.

And the really great thing is that you can make profile changes that take effect immediately without having to leave the HCD dialog!

For more information about setting HCD profile options, see the “Working with the HCD Profile Options dialog” topic in z/OS Hardware Configuration Definition (HCD) User’s Guide, SC33-7988.
Many large System z installations currently rely on Intersystem Cluster Bus (ICB) coupling links to support their Parallel Sysplex configurations. Because there is no support for ICB features on post System z10™ servers, you must migrate to the new Parallel Sysplex InfiniBand (PSIFB) coupling links before acquiring a post System z10 server. The good news is that PSIFB is working well on growing numbers of System z configurations and is ready for even the most demanding data sharing workloads!

Migration to PSIFB requires proper planning to achieve the best results. To help you understand PSIFB and its migration implications, this article discusses the following topics:

- Understanding the industry standard InfiniBand
- Understanding how System z implements InfiniBand as PSIFB
- Planning considerations for PSIFB
- PSIFB questions and answers
- Moving to PSIFB.

**Understanding the industry standard InfiniBand**

InfiniBand is an industry standard interconnect architecture that leverages switched, point-to-point channels, or both with theoretical bandwidths of up to 12 GB per second (GBps) and is designed to carry multiple traffic types such as clustering, communications, storage, and management.

InfiniBand addresses challenges that IT infrastructures face as ever-increasing demands for computing and storage resources are placed on the architecture. Specifically, InfiniBand has several advantages:

- Superior performance: InfiniBand provides superior latency performance and products supporting up to 120 GBps connections.
- Reduced complexity: InfiniBand allows for consolidation of multiple I/Os on a single cable or backplane interconnect, which is critical for blade servers, data center computers and storage clusters, and embedded systems.
- Highest interconnect efficiency: InfiniBand provides efficient scalability of multiple systems, providing communication processing functions in hardware—relieving the CPU of this task.
- Reliable and stable connections: InfiniBand provides reliable end-to-end data connections.

**Understanding how System z implements InfiniBand as PSIFB**

System z servers take advantage of InfiniBand technology in several ways:

- **I/O interface:** InfiniBand, which includes the InfiniBand Double Data Rate (IB-DDR) infrastructure, replaces the Self-Timed Interconnect (STI) features found in prior System z servers.
- **Interconnect for Parallel Sysplex:** using the following technology:
  - PSIFB links, based on 12x InfiniBand technology, support point-to-point connections up to 150 meters (492 feet) and can replace both ICB-4 and ISC-3 links.
  - PSIFB Long Reach links, based on a 1x InfiniBand technology, support point-to-point connections of up to 10 km (6.21 miles) unrepeated or up to 100 km (62.14 miles) when repeated through a dense wavelength division multiplexer (DWDM). These links can replace InterSystem Channel (ISC-3). The PSIFB Long Reach feature is available only on System z10.
- **Server Time Protocol (STP):** timekeeping signals on all PSIFB links as they are for all supported coupling links.

**Summary of the advantages of PSIFB**

Simplify Parallel Sysplex connectivity by doing more with less:

- Share physical links by defining multiple logical links using channel path identifiers (CHPIDs)
- Consolidate multiple legacy links (ISC or ICB, or both)
- Address link constraints (for instance, define another CHPID to increase available subchannels instead of adding physical links)
- Have more flexible placement of systems in a data center:
  - InfiniBand coupling links (FC 0163 and 0167) take advantage of optical cables up to 150 meters (492 feet), removing the restriction of 7 meters (22 feet) between System z central electrical complexes.

**Planning consideration for PSIFB**

IBM intends for System z10 to be the last server to support ICB-4 links (IBM hardware announcement 108-154). Clients must migrate to InfiniBand coupling before acquiring a post-System z10 server.
As migration to PSIFB takes significant planning and can be complex, care must be taken to understand environmental needs to achieve best results. A PSIFB coupling implementation must be carefully planned within the overall plan for implementing System z10 (and beyond) servers. Introducing System z10 and PSIFB into an existing environment requires a thorough understanding of the coupling technologies involved. A conservative approach that minimizes disruption is likely to be a multi-step process.

The PSIFB technology was introduced with System z10 servers and made available for System z9® servers. Therefore, implementation of PSIFB coupling always follows a System z10 server installation or the installation of a PSIFB feature on a z9® server, either as a step in the initial implementation plan, or later as a follow-on activity. Planning information is summarized in Figure 1.

### Moving to PSIFB

PSIFB can be used to reduce coupling infrastructure costs although these benefits are not always easy to quantify. PSIFB supports the same workload throughput as ICB4 (the older high-speed coupling link) and offers higher throughput than ISC3. Due to a minor difference in service times, there may be a small increase in coupling overhead (MIPS used to support coupling activity) with PSIFB compared to ICB4, but the savings in infrastructure costs often make this a moot point. These trade-offs should be evaluated prior to a move to PSIFB. See “Coupling Facility Configuration Options” in Find out more.

Using InfiniBand can reduce hardware link requirements as follows:

- **Allowing additional CHPIDs per physical link.** Each peer-mode coupling link supports seven subchannels or seven concurrent messages to a coupling facility. If the volume and duration of coupling facility (CF) accesses is high enough to cause subchannel busy conditions (for example, when sharing data across distances), additional subchannels can improve communication performance without using more physical links.

- **Allowing multiple sysplexes to share the same physical link.** This is useful if connecting to a server containing multiple CFs at the “receiver” end. For example, if there are two CHPIDs defined at the “sender”, one CHPID can be directed to connect to one CF, the other to the other CF. Depending on the configuration, this configuration reduces the number of coupling links required.

So while the limitation of seven subchannels per CF link CHPID still exists, this can be overcome by defining multiple CHPIDs over the same link.

InfiniBand supports up to 16 CHPIDs per physical link across two ports on an HCA2-O fan out. By contrast, you can configure up to two, ICB-4s, one per port, on a memory bus adapter (MBA) fan out. (Try starting with no more than eight CHPIDs across an HCA.)

Also, coupling efficiency is the function of relative differences in processor speed between the server and CF. Coupling efficiency can be improved by not only upgrading your CF server but also by migrating from ISC-3 to PSIFB links if the servers are within 150 meters of each other.

It is important to note the maximum number of coupling links and the maximum numbers of CHPIDs defined are both 64. Defining internal coupling (IC) links and defining multiple CHPIDs over installed PSIFB ports might cause the limit on the number of CHPIDs to be reached well before the maximum number of physical links. Although there is no change to the maximums, you must consider them when planning to use the flexibility available from PSIFB.

---

**Figure 1. Planning considerations for System z**

- **PSIFB ISC-3, ICB-4**
- **12x1B-SDR**
- **1x IB-DDR 10 km unrepeated**
- **12x IB-DDR up to 150m**
- **z990 ISC-3, ICB-4**
- **z890 ISC-3, ICB-4**
- **z10 PSIFB ISC-3 and ICB-4 (CFCC 16)**

* Greater distances supported with qualified WDM solutions*
PSIFB is designed to meet the coupling requirements of current and future generations of System z servers. For more information, see the following Redbooks and Web site:
- IBM Redbooks *Getting Started with InfiniBand on System z10 and System z9*, SG24-7539
- [www.infinibandta.org](http://www.infinibandta.org)

STG z/OS Lab Services has gained considerable experience with PSIFB planning and implementation, and can provide fee services to assist your migration. If you need help with analyses and migration plans, contact STG Lab Services Opportunity Management team, Anthony Di Lorenzo (dilorenz@us.ibm.com, 845-926-3838), Gerald Koger (jerrykog@us.ibm.com, 623-505-4932), or send a note to stgls@us.ibm.com.

### PSIFB questions and answers

Table 1 lists some basic questions and brief answers when considering PSIFB.

<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
</tr>
</thead>
</table>
| What types of systems connect by PSIFB coupling links?                   | • There are differences in the implementations of PSIFB on System z10 and System z9.  
• The System z9 uses optical links to connect to the System z10 however; you cannot use PSIFB to connect two System z9 mainframes. |
| Have hardware and software prerequisites been satisfied?                | In addition to minimum hardware and software levels required to install and define PSIFB coupling links, there are restrictions on the types of systems that can coexist in a Parallel Sysplex with System z10 systems. For example.  
• System z10 cannot coexist in the same sysplex with anything earlier than z990  
• PSIFB requires z/OS V1R7 or later with PTFs.                                                                                  |
| What are the new adapter identifiers (AID) for PSIFB coupling links?     | • InfiniBand host channel adapters (HCAs) have a new identifier called an adapter ID (AID). You can define multiple CHPIDs over the two ports represented by the AID.  
• The new CHPID type, CIB, identifies a coupling over a PSIFB channel path.  
• PSIFB coupling link CHPIDs can share or span across logical partitions and channel subsystems on System z10 and System z9 servers. |
| What are the cabling requirements?                                       | • PSIFB coupling links require new types of cable and connectors. Find all the details in IBM Redbooks *Getting Started with InfiniBand on System z10 and System z9*, SG24-7539. |
| Does the implementation of PSIFB coupling links involve migration from previous link technologies? | • It is likely for your installation to introduce PSIFB coupling links into an existing Parallel Sysplex environment that is already coupled through other coupling link types.  
• There are changes to physical planning for the PSIFB because cabling distances extend beyond the ICB-4 restrictions.  
• PSIFB is also attractive if you have multiple sysplexes running in the same central processor complex (CPC). |

*Table 1. Questions and answers about PSIFB and InfiniBand*
The pressure on today’s data center is greater than ever. Businesses try to sustain growth while maintaining operating efficiency. As the world becomes more interconnected, demands on information technology grow and the critical need for students who can support enterprise systems (mainframes) continues to increase.

Building a smarter planet involves thinking and acting in new ways to make systems more efficient, productive, and responsive. There are four key facets to this effort:

- **Smart work**—How can we work smarter, supported by flexible and dynamic processes that are modeled for the new way people buy, live, and work?
- **New intelligence**—How can we take advantage of the volume of real-time data available from multiple sources to make more intelligent choices?
- **Green and beyond**—How do we align our goals and behaviors with our new responsibilities so that caring for our planet and its people is no longer perceived as generosity or sacrifice?
- **Dynamic infrastructure**—How do we create a technological infrastructure that drives cost down, is intelligent and secure, and is as dynamic as today’s business climate?

For an enterprise to succeed in business today, a smarter, dynamic IT infrastructure is needed with IBM System z as the cornerstone delivering the qualities of service that clients trust.

The skills needed to set up and maintain dynamic IT infrastructures will be in high demand as businesses continue to deploy smarter technologies with IBM System z. IT managers are searching for candidates who can set up and support IBM enterprise systems. Businesses continue to depend on mainframes for building a dynamic infrastructure with extreme virtualization, workload management, capacity on demand, distributed systems, and specialty engines, just to name a few.

**Students for a dynamic infrastructure**

In order to help IBM clients build a smarter classroom, the IBM Academic Initiative works closely with colleges and universities to provide better education to millions of students and produce a smarter, more competitive IT workforce. IBM assists professors and students to work smarter by making available to them connected, collaborative environments, no-charge access to IBM hardware, technology and tools (thousands of software titles), course materials and curricula, and skills enhancement supported by a worldwide community of IBM volunteers.

Students from IBM Academic Initiative member schools are developing mainframe knowledge and IT skills to help support smarter systems and applications that can rapidly respond to changing business requirements. For example, students from member schools are developing:

- Fundamental skills to administer mainframe operating systems (z/OS, z/VM®, z/Linux) that provide dynamic and virtualization resources to businesses, thus allowing for increased flexibility and optimized server utilization
- Multi-language applications in a modern integrated development environment (IDE) to drive greater efficiency through modern application development environments like IBM Rational® Developer on System z
- The knowledge needed to work on commercial databases (like DB2®) and transaction processing systems (like CICS) to increase data serving capabilities and optimize the performance and management of data.

In a major collaborative project, IBM, Syracuse University, and the State of New York are providing a great example in which students are incorporating smarter technologies into their new campus data center to reduce energy use. For more information, see the following Web site:

David Dischiave, Director of the graduate Information Management Program in the School of Information Studies at Syracuse University, and Susan Dischiave, professor in the School of Information Studies, have been instrumental in introducing smarter enterprise systems content within the curriculum and by demonstrating the economic and technological value that students will bring to their future employers.

The classes developed and taught by the Dischiaves, “Enterprise Technologies, Global Computing Challenges” and “Database Management Systems,” provide a basic understanding of the technical and management architectures that comprise large-scale computing environments.

These courses address topic areas that include:
- Performance
- Capacity planning
- Backup and recovery
- Operating system selection
- Designing and building a data center of the future
- Data center operations and infrastructure
- The symbiotic relationship between data center operations and application development
- Managing geographically dispersed data centers
- Fault tolerance
- Redundancy and data center governance
- Creating application development policies, standards and guidelines
- Software reuse design
- Software maintenance and change control
- Configuration management.

Through smarter education, we are making a difference in the health of the global economy, the health of our planet, and the health of society. And, by bringing together stakeholders and experts from across business, government and academia, we help improve our smarter planet readiness. Figure 1 shows how IBM’s Academic Initiative along with the business and academic communities is putting all these varied skills together.

For more information about schools teaching about the mainframe, see the following Web site:

ibm.com/university/systemz

IBM helps to make the planet smarter
Academic Initiative helps develop skills for a smarter planet

Skills to develop flexible and dynamic processes modeled for the new way people buy, live, and work.

Smart work

Skills to take advantage of the wealth of information available in real time from a multitude of sources to make more intelligent choices.

New intelligence

Skills to reduce energy costs, drive greater efficiencies, respond more quickly by taking action now on energy, the environment, and sustainability, and develop new portfolio of green services and products.

Green and beyond

Skills to create intelligent infrastructure that drives down cost, is secure, and is just as dynamic as today’s business climate.

Dynamic infrastructure

Figure 1. Academic Initiative helps develop skills for a smarter planet
The z/OS UNIX System Services shell submit command can easily issue job control language (JCL) statements from the shell. This article introduces the submit2 command, which goes a step farther than submit by not only submitting the JCL, but also returning the job output when the job ends.

The submit2 command is different from the submit command in the following ways:

- Submits the JCL and returns the job output when the job ends
- Requires that TCPIP is active on the target machine
- Requires /etc/ftp.data option LEVEL JESINTERFACELEVEL 2 on the target machine
- Uses File Transfer Protocol (FTP) to submit JCL into the JES input queue
- Runs in a Linux bash shell as well as a z/OS UNIX ksh shell
- Issues JCL that resides in four different sources:
  - Local shell file
  - Local MVS data set
  - Remote shell file
  - Remote MVS data set

The submit2 command is portable, which means you can run it in either in a z/OS UNIX ksh or a Linux bash shell. Figure 1 shows the submit2 command with options to submit the local file that contains the JCL.

```
submit2 localpath 'my/jcl' 9.168.192.12 USER1 password1
```

Figure 2 shows the submit2 command's remote MVS data set code that submits JCL in an MVS data set on the target remote machine. In this situation, submit2 can run in a Linux bash shell anywhere in the world and submits JCL that resides in the target machine. Note that you use ftp get to grab the JCL, switch the JES mode using:

```
site file=jes
```

Then use ftp put to submit the job to JES.

```
./submit2 remotemvs 'THIS.IS.MY.JCL(HERE)' 9.123.56.7 USER2 password2
```

How does submit2 retrieve the job output after the jobs ends? First, submit2 must find the job and its status. Figure 3 shows the code to find the job by the job name in the JES queue.

```
FOUND0=$(cat $JCLLISTOUT|grep $JOBID)
FOUND=$(echo $FOUND0|cut -f $CUTFOUND -d ' ')
```

Figure 4 shows how the ftp get command gets the output from the JES output queue by JOBID.

```
ftp -v -n $INHOST << SCRIPT
user $INUSER $INPSWD
get $JOBID
quit
SCRIPT
}
```

When the status of the job is OUTPUT, it means the job is finished. Now it’s time to retrieve the job output. Figure 4 shows how the ftp get command gets the output from the JES output queue by JOBID.

You can download submit2 from the z/OS UNIX Tools and Toys Web site:

[ibm.com/servers/eserver/zseries/zos/unix/tools]
Does your SMB need a tune-up?

Tuning up SMB

BY KARL LAVO

That great idea you had to use the z/OS Distributed File Service SMB server to allow Microsoft® Windows® and Linux users access to z/OS UNIX file systems as network drives was a big hit at your installation. But this morning you overheard rumblings about the response time for storing and retrieving documents.

This situation is similar to what happens when you start a new computer and try out all the bells and whistles. It runs so fast it’s like cruising the internet with a rocket. Yet over time, things change. Before you know it, the computer slows to a crawl and you are texting between downloads. The same thing can happen as you use the z/OS Distributed File Service SMB server.

In this case, it isn’t an issue of malware or a spyware infiltrating your computer. As the z/OS Distributed File Service SMB server runs, it captures all sorts of resource and performance information. For example, when you open a file, the client sends the request to the server and from there, the process of opening the file quickly becomes complicated. The server moves through various caches and data structures to process your request, then calls z/OS UNIX System Services to obtain the file and access the data. There is also the communication layer and network traffic associated with this request. All this complexity might be what’s causing those rumblings you heard this morning.

No need to panic—help is on the way! Let’s take a quick look at the options and information available and possibly tweak your configuration.

SMBCOMM query report

Figure 1 shows an example of the SMBCOMM report, which displays the asynchronous communication layer of the SMB server. Essentially, it’s a snapshot of the SMB server traffic. It shows statistics of the asynchronous communication socket calls handled by the file server and the amount of queuing that occurs from handling the client requests. With asynchronous communications, the server is able to process requests and receive data requests from one client, while it processes connection requests and handles data requests from other clients.

In this report, we can see that the number of SRB requests queued is 6784. And the percentage queued is 85.9%. That doesn’t bode well for response time—no wonder you heard some rumblings this morning!

How to improve this scenario

You can control the number of primary processing threads available to the server by specifying a value for the _IOE_SMB_MAIN_POOL_ environmental variable in the DFSKERN file, located in the /opt/dfdlocal/home/dfskern directory. Pool: 0 in Figure 1 shows the number of primary processing threads available, which is the default of 14. You can increase this value to make more threads available to handle SMB requests. Keep in mind that after you change the DFSKERN file, you must then restart the server.

As with any system setting, you must be aware that increasing the number of threads does have system implications. In this case, the setting is restricted to the maximum number of threads available.
**REPORTS TO TUNE BY**

**z/OS Distributed File Service SMB** provides a QUERY command that displays information to help you determine where time is spent and identify bottlenecks. The type of information you see depends on which type of report you select.

The syntax for the query command is:

```plaintext
modify dfs,send dfskern,query,<REPORT>
```

You can select one of the following reports specific to the z/OS Distributed File Service SMB server:

**ADAPTERS**
This report displays storage use and performance information. It includes:
- File status
- Directory contents
- File name lookup information
- File security cache used by the physical file system (PFS) and record file system (RFS)
- Number of calls made to the PFS and RFS
- Average response time of each call.

**ALL**
This report shows every report that the query command can generate.

**FILESETS**
This report provides statistics regarding file sets that the server used. It displays the following information:
- File set name (file system)
- Aggregate ID
- Status flag field (A = attached, M = mounted, E = exported)
- The number of SMB servers that accessed the file system.

**LEVEL**
This report provides the service level of the SMB server.

**LOCKING**
This report displays lock facility statistics and includes:
- Number of lock waits
- Average lock wait time
- Time threads sleep (waiting for certain events to complete).

**RFS**
This report shows various statistics, access method counts and response times if you are exporting record file system datasets to your window clients.

**SERVTHREADS**
This report queries the server and returns status of currently running service threads.

**SESSFS**
This report displays active SMB session and file system information.

**SMBCOMM**
The report shows the TCP/IP asynchronous socket calls made by the file server for SMB client communications and the amount of queuing for incoming SMB requests.

**SMBINT**
The report shows statistics pertaining to the following processes:
- Number and type file requests made to the server
- Average file server response and CPU time
- TCP/IP send and receive time that you can use to determine the network overhead for file server response time.

**SMBMAP**
This report queries the current SMB map table and displays the user mapping between the SMB user to z/OS user.

**SMBPRT**
This report provides statistics of calls made to the InfoPrint server(s).

**SMBSESS**
The report returns active SMB session information including:
- Computer name
- Number of open files
- Socket number.

**SMBUSERIDSHARES**
This report returns information about the export of automounted file systems.

**SMTKC**
The report displays storage usage and performance information pertaining to the SMB tokens and the token cache generated from client requests.

**THREADS**
This report queries and returns the status of currently running threads.

**TKM**
This report returns token manager statistics for the SMB file server.

**VM**
This report queries and displays the server statistics for virtual memory caching.

**VNODE**
This report queries and displays vnode information. A vnode represents a file system object such as a directory or file.

Find additional information and full descriptions of these reports in **z/OS Distributed File Service SMB Administration Guide**, SC24-5918-09.
Introduced in 1994, SAP (pronounced “sap” as in the tree) on IBM System z offers highly available and flexible business solutions by combining reliable computer systems with standard business software. The SAP solution on DB2 for z/OS stands for data integrity, high volume transactional computing, and near continuous availability!

The challenge: SAP on DB2 and the test environment

However, to ensure that SAP on DB2 for z/OS remains the perfect choice for the broad range of SAP customer scenarios requires a comprehensive test environment with a large amount of diverse test systems, each one built from DB2 for z/OS subsystems, servers, software stacks, and full functioning SAP systems on the top. But the high volume of these test systems and their frequency of provisioning and de-provisioning demand an efficient way to deploy and optimize IT resources.

IBM and SAP development teams

The joint IBM and SAP development team responsible for the SAP on IBM System z solution understands the growing demand and timely availability of test environments. Similar requirements have also been coming from independent SAP and IBM test teams that participate in the SAP solution verification and product rollout.

Instead of addressing these growing requirements by throwing in more hardware and costly people resources, the development team has chosen a smarter approach by providing complete SAP test environments on demand and just in time!

Here’s how we responded to the challenge.

The response: data provisioning and more

In the good old days when SAP was delivering ‘just R/3’ we had a pretty stable IT environment. Users knew their own DB2 subsystems and servers like the backs of their hands. The installation and maintenance of those resources were manual processes. The database administrator (DBA) had to review every request.

Then, SAP came up with the NetWeaver solution for Java and its corresponding Business Suite, topped by the introduction of Unicode. All this dramatically increased the workload for the DB2 database and z/OS administrators.

In 2003, our test environments and z/OS database instances had doubled compared to the year before. Not enough, more sophisticated SAP test processes required multiple backups of the system in order to clone test environments. Along with new DB2 releases optimized for SAP more frequent DB2 administration tasks (like installing PTFs) had to be applied during the test process. But the IT staff did not grow, and suddenly, DB2 administration had become the major bottleneck of the SAP on System z rollout process!

Database provisioning system for DB2 on z/OS is born

Automation was our answer to this challenge. We analyzed the requirements and designed an automation solution addressing our issues.

This database-provisioning component automates the most important database administration tasks. It runs on z/OS UNIX System Services and provides these tasks as services for the DBA:

- Installation of a DB2 subsystem
- Backup of a DB2 subsystem
- Restore of a DB2 subsystem
- Cloning content of a backup into a DB2 subsystem
- Application of new service level to a DB2 subsystem.

Additionally, these tasks were made available as services to the teams.

All of this made the DBA happy because the database-provisioning component took over the routine tasks. Now, the DBA could concentrate on the really important things again.

End users were happy too because of the dramatically reduced request and response times that resulted from
database provisioning. Here are just two of the benefits:

- Elimination of the detour through administration personnel
- Ability for authorized users to perform DBA tasks and operations.

But does the DBA lose control over database activities? Not at all!

The DBA grants rights to execute a certain set of requests to the end user and might modify or revoke them according to IT administration requirements. The DBA together with the z/OS system programmer provides the necessary definitions within z/OS (for example, RACF, HSM). They still control the resources. It’s a well-defined task sharing between the automation solution and the IT administration.

How provisioning works
A set of batch jobs with additional pre- and post-execution steps defines every service. A workflow file declares the order and expected condition codes. The database provisioning component prepares the batch job, which is mainly substitution of variables, and takes care of running the job. DB2 for z/OS stores the workflow definition itself and all necessary database parameters, like the DDF port and location, SCCSID, and so on.

How to exploit DB2 for automation
DB2 delivers the means for automation. Just set up a set of DB2 stored procedures, which by the way are also required to run SAP. We explicitly exploit:

- ADMIN_JOB_SUBMIT
- ADMIN_JOB_QUERY
- ADMIN_JOB_FETCH
- ADMIN_JOB_CANCEL.

(For an excellent description of these procedures, see the Redbooks publication reference at the end of this article.)

Overcoming obstacles
During the implementation, we found some minor issues that we’d like to share with you.

The substitution of parameters isn’t a big deal. However, because of the nature of a batch job (remember the old punch cards?), it can become tricky. Take care handling continuation characters in the Job Control Language (JCL) statements (for example, in JCL DSNTJUZ at position 72).

Some of the batch jobs have to run on the same LPAR as the DB2 database, while others can run wherever it’s best in the Parallel Sysplex. We store the LPAR name for every DB2 database as part of our metadata and substitute the corresponding parameter in the JCL template:

/* JOBPARM SYSAFF=#LPAR# */

After the batch job runs, be sure to check its status by calling ADMIN_JOB_QUERY as long as the status is not OUTPUT.

Finally, you have to decide whether the batch job did what you asked. This is a matter of comparing actual with expected condition codes. Unfortunately, no procedure currently exists to get condition codes automatically. Retrieve the batch job output (ADMIN_JOB_FETCH), and parse for message “IEF142I” for every step. Figure 1 provides an overview of database-provisioning in a SAP environment:

Database pools and user groups
Together with this database-provisioning component, we also introduced the concept of database pools and user groups into our IT operation. These allow us to provide services to different groups or departments. Each group has access only to its own database pool, so we’ve separated pools from each other.

Driving more requests
Now, after five years of continuous operation, the database provisioning component has driven more than 15,000 automated requests. Several thousand DB2 installations and almost 1,300 database cloning tasks were executed, and we now manage more than 150 DB2 subsystems.

Today, the database-provisioning component with its DB2 administration services is indispensable for the SAP test teams in their daily work. Because database provisioning simplifies the process, people without System z, z/OS, or DB2 skills are now able to participate in SAP testing for System z.

Besides the enormous improvements gained with the introduction of the database-provisioning, growing complexity of SAP system landscapes led us to address further automation demands.

Full provisioning of complete ready-to-run SAP test systems
For full provisioning of SAP test system on System z, database automation had to be supplemented by provisioning.
means of the SAP application tier. To achieve this, we developed automation solutions for server and operating system provisioning (for example, Linux on System z), including SAP application server provisioning.

For server and operating system provisioning, we used a component developed by the IT Testlab Böblingen based on the Tivoli Provisioning Manager (TPM). By developing the missing TPM-based SAP application provisioning system, we put in place all the building blocks for a fully automated SAP System provisioning on System z with DB2 for z/OS in the center role.

On-demand service composer is born

Following the expectation of the end users for a single automation solution, rather than dealing with a collection of individual automation components, we realized that a coupling of those components at a higher level was necessary.

That coupling must include services of the underlying components to power workflows offered through an intuitive Web interface.

The service composer runs on z/OS UNIX as a web application acting as a lightweight enterprise service bus. It provides:

- Web service registry
- XML-based service composition framework
- Workflow engine
- Web-based graphical user interface (GUI)
- Role-based user management.

The service composer achieves service composition by processing XML-based workflow definitions and by utilizing Web services that the automation components provide.

Now users who need resources or complete SAP systems can request them through the web interface by invoking the appropriate workflow. The service composer performs the complex task of building complete SAP systems. It offers ready-to-run SAP systems to the SAP on System z teams in development, test, consulting, and pre-sales.

The System z Test Cloud for SAP on DB2 for z/OS enters the scene

The emergence of the System z Test Cloud has occurred through evolutionary steps.

Our automation efforts have demonstrated how System z fits into the IBM Cloud strategy. It helps us build smarter IT solutions with less complexity while simultaneously leveraging the strengths of System z: high availability, reliability and security.

Figure 2 shows a System z Test Cloud for SAP on DB2 based on this on-demand concept:

z/OS-based solutions reign supreme for SAP!

The provisioning and service composition technologies developed for the IBM System z platform and deployed in the SAP on the IBM System z platform team might come with products delivered through the IBM strategic dynamic infrastructure to the benefit of all z/OS customers. Having these technologies, IBM System z and the SAP solution play a significant role in IBM Cloud Computing and in the dynamic world of today’s data sharing environments.

References

DB2 9 for z/OS Stored Procedures: Through the CALL and Beyond, SG24-7604:
www.redbooks.ibm.com/abstracts/sg247604.html

Also, be sure to visit these Web sites for information:

ibm.com/systems/z/solutions/editions/index.html
ibm.com/systems/z/solutions/editions/cloud/index.html
ibm.com/servers/eserver/zseries/software/sap/
Okay, just what can the DEVSERV QL command do for you? More than you think. Here is a quick review of some of the functions!

Displaying tape libraries in the configuration

QL stands for Query Library and DEVSERV QL provides a simple way to display your tape library devices. For example, you can use DEVSERV QL, LIST to produce a list of your configuration tape libraries, as shown in the following example:

```
DS QL, LIST
IEE459I 10.03.56 DEVSERV QLIB 599
The following are defined in the ACTIVE configuration:
*CA002 *BA044 *CA030 *BA016 *BA038 *BA032 *BA031 *BA082 *B0002 B0009 BA074 BA039
```

Libraries are listed by their 5 digit library-id (libid). An libid identified with an asterisk (like *CA002) has at least one device that is physically attached to the system. An libid without an asterisk (like B009) indicates that the library has been logically defined by the Hardware Configuration Definition (HCD) but does not have any devices attached to the system.

Displaying devices and device pools for a library

To display the devices and device pools for a specific library, use the DEVSERV QL, libid command. Figure 1 shows the output from DEVSERV QL, libid command:

```
DS QL,CA002
IEE459I 10.12.50 DEVSERV QLIB 603
The following are defined in the ACTIVE configuration:
LIBID PORTID DEVICES
CA002 02 07C7* 07C0* 07C1* 07C2* 07C3* 07C4* 07C5* 07C6*
07C8 07C9 07CA 07CB 07CC 07CD 07CE 07CF
03 07D0* 07D1* 07D2 07D3 07D4 07D5 07D6 07D7
07D8 07D9 07DA 07DB 07DC 07DD 07DE 07DF
01 0780* 0781* 0782* 0783* 0784* 0785* 0786* 0787*
0788 0789 078A 078B 078C 078D 078E 078F
```

In Figure 1, devices that are physically connected to the system are displayed with an asterisk.

Devices in a library are grouped into one or more device pools, each identified by their PORTID. So what exactly are device pools? Device pools are simply a way to group library drives for allocation. For device allocation, a library pool is used much like an esoteric; that is, a library pool represents a list of devices with equivalent capability. During initialization, the system obtains a list of all of the library device pools and uses that information to build one or more eligible device tables (EDTs) for device allocation.

ACTIVE and NONACTIVE tape library configurations

Did you notice in Figure 1 that the configuration display indicates that it is for the ACTIVE configuration? The allocation process makes use of two tape library configuration views, the ACTIVE configuration and the INACTIVE configuration. After allocation has built the EDTs, the library configuration is “frozen” and becomes the ACTIVE configuration. The ACTIVE configuration indicates device use by all system components and always matches the allocation EDTs. The system makes a copy of the active configuration, and this copy is known as the INACTIVE configuration.
A problem scenario
Let's suppose you attach a device, 0180, that has not been defined in the configuration to the system and not included in an EDT. Because it is not defined in an EDT for the configuration, when you VARY device 0180 ONLINE you cannot allocate it. That is where the INACTIVE configuration comes in. The system fails the VARY ONLINE but adds device 0180 to the INACTIVE configuration with the following message:

IEA437I TAPE LIBRARY DEVICE(0180), ACTIVATE IODF=xx, IS REQUIRED

HCD to the rescue
Why must you activate an IODF? It all has to do with those pesky EDTs. When you do an IODF activate, allocation rebuilds the EDTs using the INACTIVE configuration that now includes device 0180. At that point, the INACTIVE tape library configuration becomes the new ACTIVE configuration, and the system makes a new INACTIVE copy from it.

You can avoid having to activate the IODF in this situation by using HCD. Here is how it works.

Library devices are “self-defining,” which means that when they are initialized during IPL the system learns the libids and portids so allocation can build the EDTs; however, if a device (or an entire library for that matter) is not available for initialization, it is not included in the EDT definitions. Using HCD device definitions, you can add the libid and portid to your HCD configuration to allow the system to create EDTs for all devices, even those that are not connected to the system during IPL.

See your composite libraries in a whole new way
DEVSERV QL recently got better at displaying composite libraries. With OA24965 installed (z/OS V1R8, V1R9, and V1R10) and when all of the distributed libraries are at the TS7700 Release 1.5 level or later, DEVSERV QL can display either the composite library or any of its distributed libraries. In the following example composite library BA008 includes distributed libraries BA08A and BA08B. DEVSERV QL displays both the composite view of the library and the distributed view of library BA08A:

DS QL,IAE438I
IAE438I THE FOLLOWING TAPE LIBRARY DEVICES ARE UNAVAILABLE:
7E30  7ED0  7DF0  7D50  7E90  7DB0  7DD0  7D30
7E50  7EB0  7D70  7E10  7E11  7E12  7E13  7E14
7E15  7E16  7E17  7E18  7E19  7E20  7E21  7E22
7E23  7E24  7E25  7E26  7E27  7E28  7E29  7E30
Legend:
06- Device was unavailable to the Library Manager

What else can QL do for you?
Ever see an IAE438I message? They are easy to miss. IAE438I is issued during initialization as an informational message, so it’s easy to overlook in the SYSLOG. The message looks like this:

IAE438I THE FOLLOWING TAPE LIBRARY DEVICES ARE UNAVAILABLE:
7E10  7E11  7E12  7E13  7E14  7E15  7E16  7E17
7E18  7E19  7E20  7E21  7E22  7E23  7E24  7E25
Legend:
06- Device was unavailable to the Library Manager

IEA438I lists any library drives that could not be initialized during initialization. That’s the good news. The bad news is that it doesn’t tell you what caused the device to be in the list. If you attempt to VARY a device from the list ONLINE, the VARY will fail with an IAE437I message that describes the problem.

When you have OA28634 installed (z/OS V1R9 and later) there is an easier way. The DS QL,IEA438I command displays all of the unavailable devices and indicates why they are unavailable. Figure 2 shows sample output from the command:

Figure 2. Sample output from the DS QL, IAE438I command

The number following the dash in the device id (for example, 0180-06) identifies the error that was detected during initialization. Following the device list is a description of each error that appears in the list under Legend. In the above example, your CE was probably working on the string of drives when you initialized the system.

Last but not least
One last word about DEVSERV QL. Like most DEVSERV commands, if you’re not sure of the syntax, issue:

DS QL,?

For another helping of information about DEVSERV, see z/OS MVS System Commands, SA22-7627.
Do you want more power in controlling storage management with your Language Environment® for AMODE 64 preinitialization application? Do you also want more power in message handling?

Good news! You can start getting more power by specifying GETSTORE, FREESTORE, or MSGRTN service routines in the CELQPIPI service and writing your own storage or message routines to be run in the preinitialized environment. In z/OS V1R11, enhancements to AMODE 64 Language Environment preinitialization (PreInit) allow the service routine vector to support user-supplied GETSTORE, FREESTORE, and MSGRTN service routines.

Language Environment offers preinitialized environments for running applications to improve performance. The advantage of using PreInit is that it sets up a one-time environment for multiple runs of main routines and subroutines, allowing a better resource management and a faster response to meet the needs of the application.

When Language Environment added AMODE 64 support back in z/OS V1R6, support was added for preinitialized environments (CELQPIPI), but there was no support to allow the user to replace service routines similar to what is supported in the 31-bit Language Environment’s CEEPIPI interface. In z/OS V1R9, Language Environment added support to allow the user to replace the LOAD and DELETE routines. In z/OS V1R11, support is now available for the user to replace the GETSTORE and FREESTORE routines, as well as MSGRTN.

When Language Environment needs to obtain storage in the application’s key for use by the application, the user-supplied GETSTORE service routine is called. This includes both above-the-bar and below-the-bar (above and below the 16 MB line) storage. For above-the-bar storage, additional memory object attributes can be included on the call, such as dump priority, guard size, guard location, page frame size, and the user token to associate with the memory object.

When Language Environment needs to free the storage that GETSTORE obtains, the user-supplied FREESTORE service routine is called. Also, for above-the-bar storage, Language Environment can call FREESTORE to free individual memory objects. Language Environment can also call FREESTORE to free a set of memory objects associated with a particular user token that was previously provided to GETSTORE.

You might want to consider taking advantage of the support for GETSTORE and FREESTORE service routines when the application’s environment has alternative storage allocation mechanisms available for use. In addition to being able to modify the attributes of the storage the application is obtaining, you might use this support to monitor your application’s storage.

A user-supplied MSGRTN allows error messages to be processed by the application, instead of using C I/O interfaces for writing the message to stderr. The MSGRTN service routine support might be useful when the application’s environment does not tolerate file I/O.

If you are interested in managing and controlling storage or messages in the 64-bit environment instead of using default routines, check out z/OS Language Environment Programming Guide for 64-bit Virtual Addressing Mode, SA22-7569, for further details.
Our contributors

Kimberly Bailey is a Software Engineer for IBM in Raleigh, North Carolina. She is a designer for z/OS Communications Server and focuses on network management.

Jonathan Barney, CISSP®, is an IT Specialist with Systems and Technology Group (STG) Lab Services. He works extensively on Tape Encryption customer implementations, and z/OS security solutions including PKI, TKE, and hardware cryptography.

Fernando Barsooba is a Software Engineer for z/OS Communications Server Support at IBM in Research Triangle Park. His expertise includes distributed networks and general networking. He holds an MSCS from North Carolina State University and has filed two patents related to social networking.

Stephen M. Branch is an IBM Senior Software Engineer at the Silicon Valley Lab whose 25 year career has dealt with all aspects of DFSMS™. He is the author of the Catalog Search Interface (CSI). Stephen holds a Master of Science Degree from Louisiana State University.

Jay Brodfuehrer is a Senior Software Engineer at the IBM Poughkeepsie, New York, Development Lab. His expertise includes LDAP design and architecture. Jay holds a Bachelor’s Degree from SUNY University at Buffalo and has been with IBM for 26 years.

Bob Burchfield is a Senior Software Engineer working in DFSMS device support. For the past 15 years he has been responsible for device support for IBM’s automated tape libraries. He created DEVSERV QL as a diagnostic aid and has enhanced it several times since.

Ronald Edick is a developer in z/OS Security Middleware with over 25 years of IBM experience starting out in Burlington, Vermont, in IC fabrication. For the last 14 years he has been involved with ICSF in Poughkeepsie, New York.

Bob Fairlie gave up a not very promising pro bowling career when he realized he was not very good at it. Bob works in System SSL Development in Endicott, New York.

John Fischer has been with IBM for 20 years. For the past five years he has been working in Systems and Technology Group (STG) z/OS UNIX System Services (z/OS UNIX) System Test. Previously, John worked as a developer/function tester in z/OS UNIX, where he was the original developer, coder, and patent holder of HEAPOOLS.

Thomas H. Gnech is a Senior Software Engineer at the IBM Poughkeepsie, New York, Development Lab and currently a member of the ITDS for z/OS development team. His expertise includes Encryption and Java Security. Saheem has been with IBM for 23 years.

Saheem Granados is an Advisory Software Engineer at the IBM Poughkeepsie, New York, Development Lab and currently a member of the ITDS for z/OS development team. His expertise includes Encryption and Java Security. Saheem has been with IBM for 23 years.

Stephen Hart, CISSP®, is a Staff Software Engineer for z/OS Cryptographic Services in Poughkeepsie, New York. In his six years with IBM, he has worked on ICSF, Trusted Key Entry (TKE), Encryption Facility, and TKLM for z/OS. Currently he is the TKLM for z/OS Service team lead and works on ICSF development, design, and service.

Michael Hrencecin is the System z Platform Technical Development Team Lead for Business Continuity Solutions. His expertise spans a broad range of enterprise 1/T hardware and software technologies with a focus on Parallel Sysplex and Geographically Dispersed Parallel Sysplex™ (GDPS™).

William Johnston has experience deploying encryption key management solutions for large system installations and has educated client teams on security-related topics and best practices. He has helped design and implement test approaches for security-related elements of z/OS and other platforms like Linux and Windows XP.

Steven Kocik is an Advisory Software Engineer in the System z Operating Systems Development organization in Poughkeepsie, New York. Steven is the lead developer for FICON DCM and has been with IBM for six years.

Steffen Koenig is a Software Developer with IBM for six years and works in SAP on System z Development at the IBM Boeblingen Lab. His expertise includes object oriented analyses and design, development in C++ and Java, Java 2 Platform, Enterprise Edition, Web 2.0 Technologies, SAP on System z Development, and IT Infrastructure optimization and automation.

Karl Lavo is a Software Engineer working on the DFS™/SMB development team in Poughkeepsie, New York. Karl has been involved in service and development roles for over ten years on the distributed computing architectures, which include DCE, DFS, SMB, and zFS.

George Markouzios, CISSP®, has been with RACF Development for more than 15 years and has participated in many development projects in RACF. He has led the development teams for z/OS UNIX File System access control lists, multilevel security, password phrases, installation-defined RACF checks for z/OS Health Checker, and ID propagation.

George Ng is a System z IT Specialist for IBM Systems and Technology Group Lab Services in IBM Poughkeepsie, New York. His expertise includes z/OS Parallel Sysplex, high availability, and performance. He is also co-author of the IBM Redbooks ITSO STP Planning Guide, SG24-7280, and STP Implementation Guide, SG24-7281.

* Certified Information Systems Security Professional
Jun Ogata is a Staff Software Engineer with over 15 years of experience testing z/OS security (RACF and ICSF), and likes to be known as a RACF “entomologist.” Some of his more recent assignments include testing Multilevel Security, Federal Information Processing Standards (FIPS), and helping to achieve the Common Criteria certification. Jun strives to demystify the magic of RACF.

Michael Onghena, CISSP®, is a Senior Software Engineer in the z/OS security area at IBM in Poughkeepsie, New York, and has been with IBM for 17 years.

Kathy Pfeiffer is a Senior Software Engineer at IBM in Poughkeepsie, New York. She is the Academic Initiative System z program manager.

Don Resnik is a Senior Software Engineer at IBM in Poughkeepsie, New York. He is the IBM System z Skills leader.

Sam Reynolds is a Senior Software Engineer for z/OS Communications Server at IBM in Research Triangle Park. He has an MSCS from the University of Alabama and has been with IBM for 20 years.

Dale Riedy is a Senior Technical Staff Member in the System z Operating Systems Development organization in Poughkeepsie, New York. He is an expert in the z/OS I/O Supervisor. Dale has been with IBM for 30 years.

Ira Ringel has been in RACF Development for more than 16 years. He is currently Lead Developer on the RACF Level 3 team. Over the years, Ira has participated in many development projects in RACF with a focus on RACF manager and sysplex support.

Ben Rogers is an IT Specialist for IBM Lab Services and works with customers on business and technical security solutions. His experience includes software architecture, design, development, test, and solution implementation. He also is an Adjunct Faculty member at Marist College where he teaches an accredited online class “Introduction to the New Mainframe: Security.”

William Rooney is a Senior Technical Staff Member for the System z Operating Systems Development organization in Poughkeepsie, New York. His expertise includes I/O configuration, I/O quality of service, and z/OS storage replication. Bill has been with IBM for 31 years.

Eric Rosenfeld, CISSP®, joined IBM as a Software Engineer in 1985 and has been developing security products for 17 years. He is currently a developer for RACF and is the development product owner for z/OS Network Authentication Service.

Eric Rossmann, CISSP®, is an Advisory Software Engineer for z/OS Cryptographic Services in Poughkeepsie, New York, and has been with IBM since 1998. He has worked in several zSeries security areas, including ICSF, Encryption Facility, cryptographic coprocessor development for z/OS, and kernel device driver development for Linux on IBM System z. He is a team lead for ICSF.

Holger Scheller is a Senior Software Developer SAP on System z Development at the IBM Boeblingen Lab.

Spencer Smith is a Software Engineer for z/OS Communications Server at IBM in Research Triangle Park. He currently performs function test for TCP/IP Sysplex technologies. Spencer holds a degree in Information Science from the University of North Carolina at Chapel Hill.

Meenakshi Sripal is a software developer in the Systems and Technology Group at IBM in Poughkeepsie, New York. She is pursuing a Masters Degree in Computer Science from Columbia University.

Bruce Wells is a senior Software Engineer for the RACF design and development group in Poughkeepsie, New York, with experience in both z/OS and z/VM. He has worked on projects spanning other security components (LDAP, ICSF, and System SSL) and holds a Bachelor of Science degree from the University of Massachusetts at Amherst. He has worked in RACF for 18 plus years.

Scott Woolley has spent much of his IBM career in RACF Development focusing on encryption, auditing, and identity mapping. Scott strives to find the humor in security.

Richard Young is an IBM Certified IT Specialist with 19 years of experience with System z and 12 years with IBM. Richard is currently the z/VM and Linux for System z team lead for System z Lab Services and holds degrees in Computer Science and Business Administration. His experience encompasses such System z areas as z/OS, Parallel Sysplex, WebSphere, z/VM, and Linux for System z.

Clarifications, corrections, and announcements

Visit the Hot Topics Web site for clarifications and corrections to articles from our past issues. You’ll also find late-breaking announcements for System z and our Hot Spotlight Web articles:

ibm.com/systems/z/os/zos/bkserv/hot_topics.html

While you’re there, be sure to check the errata section for the latest changes to the following article:

Ask Mr. Catalog
by Stephen Branch

* Certified Information Systems Security Professional
Test your z/OS knowledge with our very first crossword puzzle!

ACROSS
1. Last word in the first title of an article in this issue related to the weather.
5. One of the things you protect.
7. The heart of your security.
10. Works with 20 across to secure your enterprise (hint: see the lead feature article in this issue).
12. U.S. government agency that protects investors.
14. IBM function that manages what comes “IN” and what goes “OUT” (acronym).
15. No ego at all.
16. US. City by the bay (hint: it’s not Seattle).
17. A kind of MVS supervisor block.
20. Works with 10 across to secure your enterprise (hint: see the lead feature article in this issue).

DOWN
2. An application protocol for querying and modifying directory services running over TCP/IP (hint: see the feature article in this issue).
3. It used to be called OpenEdition MVS.
4. When computer technology meets communication technology (abb.) or “You’ve got to have _.”
7. It’s not the hierarchical file system.
8. JES2 and JES3 (as primary) and IMS (as secondary) make use of this MVS service.
9. What 4 articles in this issue are all about (hint: it’s not as “CRYPTIC” as you think).
11. Key security activity and a theme to this Hot Topics issue.
13. When a product or function is removed or withdrawn (acronym).
18. Two word abbreviation that insists you want it right away.
19. A name for cryptographic protocols that provide security for communications over networks (hint: see the lead feature article in this issue).

* Answers will appear on our Hot Topics Web site: ibm.com/servers/eserver/zseries/zos/bkserv/hot_topics.html
any people regard social networking tools such as Twitter and Facebook as frivolous pastimes that couldn’t possibly have relevance to their day job. However, we at z/OS Communications Server (like a number of other IBM product areas) believe that focused use of these tools can help improve our customers’ experience with our product, and ultimately, even the product itself. In this article, we will discuss why we are using these tools and why you might want to consider Tweeting along with us!

Getting on board: Companies, organizations, and government agencies enter the social networking arena

Unless you live in a galaxy far, far away, you have probably heard of Facebook and Twitter, two of the most popular social networking applications today. You may even be one of the 300 million users of Facebook, or follow breaking news through your Twitter account. Or perhaps you have remained skeptical at best about the whole social networking revolution. After all, what is the point of participating if you’re not a college student, and you don’t care that your Twitter friend is “sitting on the patio”?

Not just a virtual gathering of college students anymore, Facebook is increasingly popular in corporate America. Many major corporations, including IBM, are already using Facebook as a customer communications platform. Facebook offers one of the most targeted customer communication platforms ever created.

Twitter is another key application in the social networking world. More focused on the propagation of short messages known as tweets (text-based posts of up to 140 characters), Twitter is being used by large companies to alert people to breaking news and new features.

Even government agencies are seeing opportunities in social networking applications, using them to quickly communicate with the public, distributing information such as emergency notices.

Leveraging social media at z/OS Communications Server

z/OS Communications Server has started using social networking technologies to enhance our customer communication by providing useful product information, and soliciting customer feedback. And because social networking is a worldwide phenomenon, we’re also hoping these methods will help us reach even more of our customers.

Of course, these new social media do not replace our existing communication methods, but rather augment them.

Tweeting and getting face-to-face with z/OS Communications Server

In the summer of 2009, z/OS Communications Server established a Twitter ID called IBM_Commserver at: www.twitter.com/IBM_Commserver

From that Twitter account, we are posting periodic updates about z/OS Communications Server, with tweets on topics such as:

- Recent announcements and statements of direction
- Hints and tips
- Soliciting customer feedback, including information on how customers use certain functions.

Many of those tweets point to longer discussions on our other entry into the social networking arena: our z/OS Communications Server Facebook fan page. That fan page can be found by going to:

www.facebook.com/IBMCommserver

Please join us!

Ultimately, our success with these technologies will depend on customer participation. By joining us, you can help build a vibrant community with benefits for both our customers and our product. Please consider following us on Twitter, and becoming a fan on Facebook!