

# **AS/400 Machine-Level Programming**

Leif Svalgaard

*“And ye shall know the truth, and the truth shall set ye free (John 8:32<sup>1</sup>)*

Houston, 2001

---

<sup>1</sup> Also etched in stone in the CIA HQ’s lobby

# AS/400 Machine Level Programming

## Table of contents

Chapter 0: <b>Introduction</b>	-	-	-	-	-	-	-	0-1
Why Program at the Machine-Level?								0-1
What is the Machine-Level?	-	-	-	-	-	-	-	0-1
Above and Below the MI								0-1
Old MI and New MI	-	-	-	-	-	-	-	0-1
Is MI Hard and Arcane?								0-1
What About All Those MI-Instructions?	-	-	-	-	-	-	-	0-2
Small Programs or Large Programs								0-2
Why is MI so Secret?	-	-	-	-	-	-	-	0-2
 Chapter 1: <b>Getting Your Own MI-Compiler</b>								1-1
MI-Compilers								1-1
Create Program (QPRCRTPG) API	-	-	-	-	-	-	-	1-1
Detailed Explanation of Each Parameter (Boring)								1-2
The MI-Compiler Front-End	-	-	-	-	-	-	-	1-4
The CRTMIPGM Command								1-7
Hello World	-	-	-	-	-	-	-	1-8
MI Functional Reference Manual								1-10
 Chapter 2: <b>Data Types and Your First Real MI-Program</b>								2-1
Data Types								2-1
Character Data Type	-	-	-	-	-	-	-	2-1
Numeric Data Types								2-2
Data Names	-	-	-	-	-	-	-	2-2
Comments								2-3
Hexadecimal Constants	-	-	-	-	-	-	-	2-3
Convert Hexadecimal to Character								2-3
Compile Errors	-	-	-	-	-	-	-	2-4
Conditions and Branching								2-4
Labels and Branch Points	-	-	-	-	-	-	-	2-5
Short Form of Instructions								2-5
Structured Data	-	-	-	-	-	-	-	2-5
Our First "Real" MI-Program								2-7
 Chapter 3: <b>Pointers, Pointers, and Pointers</b>								3-1
Space Data Object								3-1
Setting a Space Pointer	-	-	-	-	-	-	-	3-2
Explore Pointers: Materialize Invocation Stack								3-2
Static Storage Method	-	-	-	-	-	-	-	3-2
Dealing with Exceptions								3-4
Automatic Storage Allocation	-	-	-	-	-	-	-	3-5
Instruction Pointers								3-6
 Chapter 4: <b>Arithmetic and Timestamps</b>								4-1
Timestamps								4-1
Convert Timestamp to Date and Time	-	-	-	-	-	-	-	4-1
Converting to a Date								4-2
Complete Program to Show Current Date and Time								4-3
Leap Seconds								4-4
Higher Resolution in Later Versions	-	-	-	-	-	-	-	4-5

<b>Chapter 5: Calling Other Programs</b>	-	-	-	-	-	-	-	5-1
External Program Objects								5-1
Obtaining a System Pointer	-	-	-	-	-	-	-	5-1
Initializing a System Pointer								5-1
Resolving a System Pointer	-	-	-	-	-	-	-	5-1
Copying a System Pointer								5-2
System Entry Point Table (SEPT)	-	-	-	-	-	-	-	5-2
Counterfeiting a Pointer								5-2
Encapsulated Objects	-	-	-	-	-	-	-	5-2
The Argument List								5-2
Program Call Example	-	-	-	-	-	-	-	5-3
Relative Branch Conditions								5-4
An Optimization	-	-	-	-	-	-	-	5-5
The System Entry Point Table								5-5
 <b>Chapter 6: RISC Code for Setting a Pointer</b>	-	-	-	-	-	-	-	 6-1
Set Space Pointer From Pointer								6-1
Documentation About the PowerPC	-	-	-	-	-	-	-	6-1
AS/400 RISC PowerPC Machine Architecture								6-1
System Service Tools	-	-	-	-	-	-	-	6-2
Dissecting SETSPFP								6-4
LQ - Load Quadword Instruction and Pointer Tag Bits				-	-	-	-	6-4
TXER Instruction								6-5
ORI - OR Immediate Instruction	-	-	-	-	-	-	-	6-6
ADDI - Add Immediate Instruction								6-6
RLDICL Instruction	-	-	-	-	-	-	-	6-6
Condition Register								6-7
BC - Branch Conditional Instruction	-	-	-	-	-	-	-	6-7
BLA - Branch with Link to Absolute Address Instruction								6-8
Link Register	-	-	-	-	-	-	-	6-8
Addresses, Segments, Offsets, and SLIC								6-8
LD - Load Doubleword Instruction	-	-	-	-	-	-	-	6-9
RLDICR Instruction								6-9
SETTAG Instruction	-	-	-	-	-	-	-	6-10
STQ, Store Quadword Instruction								6-10
Final Annotated SETSPFP Code	-	-	-	-	-	-	-	6-10
 <b>Chapter 7: Accessing Arbitrary Data in Memory</b>	-	-	-	-	-	-	-	 7-1
Accessing Arbitrary Data								7-1
Set Space Pointer From Any Pointer	-	-	-	-	-	-	-	7-1
Counterfeiting the Pointer								7-1
Accessing Encapsulated Program Code	-	-	-	-	-	-	-	7-2
Using the Debugger								7-3
System Domain Objects	-	-	-	-	-	-	-	7-4
Going System State								7-5
This Does Not Work on a CISC Box	-	-	-	-	-	-	-	7-5
 <b>Chapter 8: Getting/Setting File Member Information</b>	-	-	-	-	-	-	-	 8-1
File Member Information								8-1
File Member Cursors	-	-	-	-	-	-	-	8-1
The Member Header								8-1
Converting Dates to Timestamps	-	-	-	-	-	-	-	8-3
Must be System State								8-3
Parameters	-	-	-	-	-	-	-	8-3
Command CHGMBR and Command Processing Program CHGMBRCL								8-4
Command Processing Program	-	-	-	-	-	-	-	8-5

The Complete MIMBRINF Program									8-6
<b>Chapter 9: The Work Control Block Table</b>	-	-	-	-	-	-	-	-	9-1
Tasks, Processes, and Jobs									9-1
Job Names	-	-	-	-	-	-	-	-	9-1
Finding Work Control Block Tables									9-1
The Process Communication Object	-	-	-	-	-	-	-	-	9-1
The Master (Root) WCB Table List									9-2
Scanning the WCB Tables	-	-	-	-	-	-	-	-	9-3
Testing if a Pointer is Null									9-3
Searching a Work Control Block Table	-	-	-	-	-	-	-	-	9-3
Scanning WCB for Device Name									9-4
Checking the Job Status	-	-	-	-	-	-	-	-	9-4
Putting It All Together									9-5
Performance	-	-	-	-	-	-	-	-	9-7
A Faster Way of Locating Jobs									9-7
Format of the Job Index Header	-	-	-	-	-	-	-	-	9-7
Format of the Job Index Entry									9-8
Addressing the Job Index	-	-	-	-	-	-	-	-	9-8
Finding Entries in the Job Index									9-9
A Faster Version, MIWCBFND	-	-	-	-	-	-	-	-	9-9
A Brief History of WCBT Problems									9-11
<b>Chapter 10: Internal Sorting, Combsort</b>	-	-	-	-	-	-	-	-	10-1
Sorting Internal Data									10-1
Bubble Sort	-	-	-	-	-	-	-	-	10-1
Combsort									10-2
MI-Version of Combsort	-	-	-	-	-	-	-	-	10-2
Override Program Attribute									10-3
The Sort Double Loop	-	-	-	-	-	-	-	-	10-3
Complete Combsort Code									10-4
Using Instruction Pointers	-	-	-	-	-	-	-	-	10-4
Testing Combsort									10-5
Generating Random Numbers	-	-	-	-	-	-	-	-	10-5
Measuring Processor Time Spent									10-5
Calling Combsort	-	-	-	-	-	-	-	-	10-6
Computing $N \log_2 N$									10-6
Compute Mathematical Function with 1 Argument	-	-	-	-	-	-	-	-	10-7
Computing Time Differences									10-7
Simple Numeric Editing	-	-	-	-	-	-	-	-	10-8
The Complete MITSTCMB Test Program									10-8
Performance Results	-	-	-	-	-	-	-	-	10-10
RPG Version of COMBSORT									10-11
<b>Chapter 11: The Machine State Register.</b>	-	-	-	-	-	-	-	-	11-1
Machine State Register (MSR)									11-1
Privilege Level	-	-	-	-	-	-	-	-	11-2
The MFMSR/MTMSRD Instructions									11-2
Running in Supervisor Mode	-	-	-	-	-	-	-	-	11-3
Finding the SLIC Code for Call Program									11-3
Retrieving Machine Registers	-	-	-	-	-	-	-	-	11-5
Getting General-Purpose Registers									11-5
Getting Special-Purpose General-Use Registers	-	-	-	-	-	-	-	-	11-6
STD Instruction									11-7
Getting the Machine State Register	-	-	-	-	-	-	-	-	11-8
The Program State									11-8

The Floating Point Bit	-	-	-	-	-	-	-	-	11-8
<b>Chapter 12: Input/Output in MI Using the SEPT</b>	-	-	-	-	-	-	-	-	12-1
A File Compressor/Decompressor									12-1
The User File Control Block	-	-	-	-	-	-	-	-	12-1
How Do We Know This Stuff?									12-2
Setting Library/File/Member	-	-	-	-	-	-	-	-	12-2
Some AS/400-S/38 History									12-3
Control Flags	-	-	-	-	-	-	-	-	12-3
More Control Flags									12-4
The FCMPRS (File Compress) Command	-	-	-	-	-	-	-	-	12-4
Command-Processing Program									12-4
Parameter Block	-	-	-	-	-	-	-	-	12-4
The System Entry Point Table (SEPT)									12-5
Opening a File	-	-	-	-	-	-	-	-	12-6
The Open Data Path									12-7
Data Management Entry Point Table	-	-	-	-	-	-	-	-	12-9
I/O Routines									12-10
The Data Management Option List	-	-	-	-	-	-	-	-	12-10
The Data Management Control List									12-12
Filling the Input Area	-	-	-	-	-	-	-	-	12-12
Compressing the Input									12-12
Write the Compressed Records	-	-	-	-	-	-	-	-	12-13
Getting the System Name									12-14
Decompressing the Data	-	-	-	-	-	-	-	-	12-14
Compression Ratio									12-15
The Complete Program	-	-	-	-	-	-	-	-	12-15
<b>Chapter 13: File Conversion and Hashing</b>	-	-	-	-	-	-	-	-	13-1
Minimum File Transformation: COPY									13-1
Generic File Transformer Command	-	-	-	-	-	-	-	-	13-2
Translating the Data From a File									13-3
A GREP-Like Utility	-	-	-	-	-	-	-	-	13-4
A KWIC (Key Word-In-Context) Version of GREP									13-5
Calculating a File Digest	-	-	-	-	-	-	-	-	13-6
Clearing the File Before Output									13-8
Variable Length Part of the UFCB	-	-	-	-	-	-	-	-	13-8
Blocking Records to Increase I/O Performance									13-8
With/Without Level Check	-	-	-	-	-	-	-	-	13-8
<b>Chapter 14: Password Encryption on the AS/400</b>	-	-	-	-	-	-	-	-	14-1
Password Protected Access									14-1
Password Rules	-	-	-	-	-	-	-	-	14-1
Retrieving the Encrypted Password									14-1
Algorithm for the 1st Encrypted Password	-	-	-	-	-	-	-	-	14-2
Using CIPHER to DES-Encrypt									14-4
Algorithm for the 2nd Encrypted Password	-	-	-	-	-	-	-	-	14-5
Why Seven Characters Instead of Eight?									14-7
The Magic Constant	-	-	-	-	-	-	-	-	14-7
The Compleat MIENCPWD Program									14-8
The Brute-Force Attack	-	-	-	-	-	-	-	-	14-10
Passwords Stored in the Clear!									14-10
A Password Checking Utility	-	-	-	-	-	-	-	-	14-10
Real Security									14-12
<b>Chapter 15: Program Validation Value</b>	-	-	-	-	-	-	-	-	15-1

The Program Validation Value									15-1
States and Domains	-	-	-	-	-	-	-	-	15-1
Saving/Restoring Programs									15-1
Calculating the Program Validation Value	-	-	-	-	-	-	-	-	15-1
The ILE Program Model									15-2
Our TEST Program	-	-	-	-	-	-	-	-	15-2
Object Header									15-2
Program Header	-	-	-	-	-	-	-	-	15-3
The AS/400 Security Oracle									15-3
Effective Address Translation	-	-	-	-	-	-	-	-	15-4
Page Protection Bits									15-5
Writable Program Header	-	-	-	-	-	-	-	-	15-5
History Log									15-5
Program Version Table	-	-	-	-	-	-	-	-	15-6
Program MICLNPGM to Clean Patch Information and History Log									15-7
Program Maintenance Header	-	-	-	-	-	-	-	-	15-9
Module Table									15-10
Module Header	-	-	-	-	-	-	-	-	15-10
Module Version Table									15-11
Procedure Table	-	-	-	-	-	-	-	-	15-12
RISC Instructions									15-12
Module Constants	-	-	-	-	-	-	-	-	15-13
Primary Associated Space									15-13
Which Components Go Into the PVV?	-	-	-	-	-	-	-	-	15-13
Diffusion and Confusion									15-13
Program MIPGMVV Computes the PVV	-	-	-	-	-	-	-	-	15-14
Clearing the History Log on V4R4+									15-16
Digitally Signed Objects	-	-	-	-	-	-	-	-	15-16
 Chapter 16: <b>User-Defined 5250 Datastream I/O</b>	-	-	-	-	-	-	-	-	16-1
Display Files									16-1
The MIHWORLD Program	-	-	-	-	-	-	-	-	16-2
Data Management Option List									16-3
Data Management Control List	-	-	-	-	-	-	-	-	16-3
Device Control (or Name) Block									16-3
Machine Space Pointers	-	-	-	-	-	-	-	-	16-4
Datastream Format									16-4
Buffer Format	-	-	-	-	-	-	-	-	16-4
Datastream Commands									16-4
Orders	-	-	-	-	-	-	-	-	16-6
Field Control Word									16-7
Field Format Word	-	-	-	-	-	-	-	-	16-7
Screen Attributes									16-8
Explanation of the Magic Constants	-	-	-	-	-	-	-	-	16-9
 Chapter 17: <b>A Simple Screen I/O Interface</b>	-	-	-	-	-	-	-	-	17-1
Describing Screen Files									17-1
Item Introducer	-	-	-	-	-	-	-	-	17-2
Logical Attributes									17-2
Cursor Position	-	-	-	-	-	-	-	-	17-3
The Contiguous Data Block									17-3
The MITSTSCR Test Program	-	-	-	-	-	-	-	-	17-4
Command Keys									17-4
The MISCRNIO Screen Handler	-	-	-	-	-	-	-	-	17-4
File control Block and Data Management Entries									17-5
Buffer Formats	-	-	-	-	-	-	-	-	17-6

Orders									17-7
Command Key Translation	-	-	-	-	-	-	-	-	17-7
Data Content Translation									17-8
Attribute Translation	-	-	-	-	-	-	-	-	17-8
Current Cursor Position									17-8
The Screen Handler Code	-	-	-	-	-	-	-	-	17-9
Open Terminal									17-9
Close Terminal	-	-	-	-	-	-	-	-	17-9
Write to Terminal									17-9
The VERIFY MI-Instruction	-	-	-	-	-	-	-	-	17-10
The XLATE MI-Instruction									17-10
Calculating Where to Place the Cursor	-	-	-	-	-	-	-	-	17-11
Read from Terminal									17-11
Format of MDT Input Fields Returned	-	-	-	-	-	-	-	-	17-12
Finding Field With Cursor									17-13
 Chapter 18: (SCV) Supervisor Call Vectored	-	-	-	-	-	-	-	-	18-1
The SCV/RFSCV Instructions									18-1
How Is the SCV Used?	-	-	-	-	-	-	-	-	18-1
The SCV Dispatch Code									18-1
RLDIMI Instruction	-	-	-	-	-	-	-	-	18-2
Tracing and Single-Step Mode									18-3
ADDIS Instruction	-	-	-	-	-	-	-	-	18-3
SCV 10 Handler									18-4
The Blocked Instruction Flag Table	-	-	-	-	-	-	-	-	18-4
Other SCV Calls									18-6
What Did We Learn?	-	-	-	-	-	-	-	-	18-6
 Chapter 19: Calculating Archimedes' Constant, Pi	-	-	-	-	-	-	-	-	19-1
An Amazing Formula for Pi									19-1
Computing the Square Root	-	-	-	-	-	-	-	-	19-1
The MIPIPKD Program									19-2
The MIPIFLT Program	-	-	-	-	-	-	-	-	19-3
The Original S/38 Program, MIPIS38									19-4
 Chapter 20: Exception Handling	-	-	-	-	-	-	-	-	20-1
Exceptions and Events									20-1
Declaring an Exception Monitor	-	-	-	-	-	-	-	-	20-1
The Exception Identifier									20-2
Searching for Exception Descriptions	-	-	-	-	-	-	-	-	20-2
Materialize an Exception Description									20-2
Modifying an Exception Description	-	-	-	-	-	-	-	-	20-3
Monitoring Exceptions: The MIDECEXC Program									20-3
Exception General and Specific Data	-	-	-	-	-	-	-	-	20-4
Return from Exception									20-5
Signaling Exceptions: The MISIGEXC Program	-	-	-	-	-	-	-	-	20-6
Materialize Invocation Attributes									20-6
Signaling an Exception	-	-	-	-	-	-	-	-	20-7
Various Errors									20-8
Preventing Messages in the Joblog	-	-	-	-	-	-	-	-	20-9
 Chapter 21: Walking the ODP Chain	-	-	-	-	-	-	-	-	21-1
The Open Data Path									21-1
Searching Work Control Block Tables	-	-	-	-	-	-	-	-	21-1
The Data Management Communication Object									21-2
The DMQC Root	-	-	-	-	-	-	-	-	21-2

The SETSPPO MI-Instruction									21-3
A DMCQ Open Entry	-	-	-	-	-	-	-	-	21-5
Locating the ODPs									21-5
Structure of the ODP	-	-	-	-	-	-	-	-	21-6
The Open Feedback									21-6
The I/O Feedback	-	-	-	-	-	-	-	-	21-7
Data to Collect about an Active File									21-7
The MIODPWLK Program	-	-	-	-	-	-	-	-	21-8
 Chapter 22: <b>How to Generate a Truly Random Number</b>	-	-	-	-	-	-	-	-	22-1
What is a Random Number?									22-1
Use of Randomness	-	-	-	-	-	-	-	-	22-1
Bad Seeds									22-1
The Entropy Pool	-	-	-	-	-	-	-	-	22-2
Sources of Entropy									22-2
Timing Information	-	-	-	-	-	-	-	-	22-3
Entropy Pool Format									22-3
The Yield Time	-	-	-	-	-	-	-	-	22-4
Resource Management Data									22-4
Processor Utilization	-	-	-	-	-	-	-	-	22-5
Storage Management Counters									22-5
Main Storage Pool Information	-	-	-	-	-	-	-	-	22-6
Disk Storage Information									22-7
Activity Level Control Data	-	-	-	-	-	-	-	-	22-8
Can We Measure the Amount of Entropy in the Pool?									22-9
Aging the Entropy Pool	-	-	-	-	-	-	-	-	22-9
The Get Entropy Program (MIGETETP)									22-10
The Internal State	-	-	-	-	-	-	-	-	22-10
Generation of Entropy									22-11
The Entropy Monitor Program (MIETPMON)	-	-	-	-	-	-	-	-	22-12
Conclusion									22-13
TCP/IP Flaw Because of Lack of Randomness	-	-	-	-	-	-	-	-	22-13
 Chapter 23: <b>Automatic Refresh of Display</b>	-	-	-	-	-	-	-	-	23-1
The Automatic Refresh Problem									23-1
DDS Specification for the Display File	-	-	-	-	-	-	-	-	23-1
ASSUME (Assume) Keyword									23-1
CFnn (Command Function) Keyword	-	-	-	-	-	-	-	-	23-1
KEEP (Keep) Keyword									23-1
OVERLAY (Overlay) Keyword	-	-	-	-	-	-	-	-	23-1
USRDFN (User-Defined) Keyword									23-2
INVITE (Invite) Keyword	-	-	-	-	-	-	-	-	23-2
Invited Devices									23-2
General-Purpose Time-out Program	-	-	-	-	-	-	-	-	23-3
The DDS for the Invited Device									23-3
RPG Versions of the Time-Out Program	-	-	-	-	-	-	-	-	23-4
Call Stack for WRKACTJOB Command									23-4
The QDMACCIN Program	-	-	-	-	-	-	-	-	23-5
Changing the Time-out Value Dynamically									23-6
The MIINVDSP Invited Display Program	-	-	-	-	-	-	-	-	23-6
Automatic Refresh Driver Program MIAUTREF									23-8
The CVTEFN MI-Instruction	-	-	-	-	-	-	-	-	23-10
The ROLLUP / ROLLDOWN Problem									23-10
Acknowledgements	-	-	-	-	-	-	-	-	23-11
 Chapter 24: <b>Editing of Numeric Variables</b>	-	-	-	-	-	-	-	-	24-1



The Importance of Editing								24-1
Edit Codes and Edit Words	-	-	-	-	-	-	-	24-1
COBOL Pictures								24-1
The EDIT MI-Instruction	-	-	-	-	-	-	-	24-2
The Edit Mask								24-2
Parameter Length Conformance	-	-	-	-	-	-	-	24-4
Left-Justifying the Number								24-4
 Chapter 25: <b>Machine Indexes</b>	-	-	-	-	-	-	-	25-1
What is a Machine Index?								25-1
Binary Radix Trees.	-	-	-	-	-	-	-	25-1
The Index Instructions								25-1
Security Level 40+ Considerations	-	-	-	-	-	-	-	25-1
The Search Engine Problem								25-2
Index Object Subtypes	-	-	-	-	-	-	-	25-2
Creating the Index								25-2
Reading the File Member	-	-	-	-	-	-	-	25-5
Extracting Words								25-6
Inserting an Entry	-	-	-	-	-	-	-	25-7
Internal Structure of an Index								25-8
Iterating over All Members of a File	-	-	-	-	-	-	-	25-9
Insertion Speed								25-11
Searching the Index	-	-	-	-	-	-	-	25-11
Building a Screen								25-13
Searching for the Word	-	-	-	-	-	-	-	25-14
 Chapter 26: <b>The AS/400 Memory Explorer</b>	-	-	-	-	-	-	-	26-1
Examining Memory without SST								26-1
Accessing Arbitrary Memory	-	-	-	-	-	-	-	26-1
Preserving Tag Bits								26-2
Screen Layout	-	-	-	-	-	-	-	26-3
Arrays of Fields								26-3
The Explorer Program (MIEXPLR)	-	-	-	-	-	-	-	26-4
The DATA to Show								26-4
Stacking/Unstacking of Addresses	-	-	-	-	-	-	-	26-4
Showing the Screen								26-4
Enter: Specifying the Location of the Data	-	-	-	-	-	-	-	26-6
Paging Down/Up to See More								26-7
The HOME or PCS Position	-	-	-	-	-	-	-	26-7
Look at Memory at Another Address								26-7
Detecting a Pointer	-	-	-	-	-	-	-	26-8
The CMPTRT MI-instruction								26-8
Unstacking to Previous Address	-	-	-	-	-	-	-	26-9
Displaying Timestamps								26-9
Disassembling Instructions	-	-	-	-	-	-	-	26-9
Exiting the Program								26-10
Building the Screen Image	-	-	-	-	-	-	-	26-10
Determining the Pointer Description								26-11
Ready-made AS/400 Explorer	-	-	-	-	-	-	-	26-11
 Chapter 27: <b>Inside a Save File</b>	-	-	-	-	-	-	-	27-1
Why Save-Files?								27-1
The Internal Save Format	-	-	-	-	-	-	-	27-1
The Dump Space								27-2
Dump Space Management Instructions	-	-	-	-	-	-	-	27-2
Checksums								27-2

Dump Space Header	-	-	-	-	-	-	-	-	27-2
The Dump Space Data Pages	-	-	-	-	-	-	-	-	27-2
The Dump Descriptor									27-3
Dump Descriptor Catalog	-	-	-	-	-	-	-	-	27-3
Extended Object Information									27-4
Show Object in a Save File, MIOBJSV	-	-	-	-	-	-	-	-	27-4
Walk Descriptors in Save File, MIWLKSAV									27-7
Modifying a Save File	-	-	-	-	-	-	-	-	27-9
Save File Integrity? NOT									27-9
Single-Level Store? Maybe Not Always	-	-	-	-	-	-	-	-	27-10
"Flushing" a Save File to Disk									27-10
Changing Module Information, MICHGMOD	-	-	-	-	-	-	-	-	27-10
Save Files for CISC Architecture									27-13
 <b>Chapter 28: Recursion, Entropy, and the End of the World</b>	-	-	-						28-1
Recursive Procedures									28-1
Summing of Integers	-	-	-	-	-	-	-	-	28-1
Equipartition of Energy									28-2
Distinguishability of Particles	-	-	-	-	-	-	-	-	28-3
"How Many Ways to Distribute 10 Units of Energy among 5 Distinguishable Particles?"									28-3
A Computer Simulation	-	-	-	-	-	-	-	-	28-3
Programming Idioms									28-4
Storage Attributes	-	-	-	-	-	-	-	-	28-4
Calling MIADSUMR									28-5
A Parameter That Is Not a Pointer	-	-	-	-	-	-	-	-	28-6
An Iterative Version, MIADSUMI									28-6
Sample Results (Combinatorial Explosion)	-	-	-	-	-	-	-	-	28-7
Method of Finite Differences									28-8
A Note About the Programs for This Chapter	-	-	-	-	-	-	-	-	28-8
The Tower of Hanoi									28-8
Fibonacci Numbers	-	-	-	-	-	-	-	-	28-11
 <b>Chapter 29: Changing Your Machine's Serial Number</b>	-	-	-	-	-	-	-	-	29-1
What Legitimate Reason Can You Have For This?									29-1
A True Horror Story	-	-	-	-	-	-	-	-	29-1
The MATMATR Instruction									29-2
Finding the SCV Function	-	-	-	-	-	-	-	-	29-2
OS/400 is Developed Under UNIX									29-2
Contents of the Data Segment	-	-	-	-	-	-	-	-	29-2
Get the Serial Number									29-3
The HDTA Object	-	-	-	-	-	-	-	-	29-4
System Values									29-5
What Happens at the Next IPL?	-	-	-	-	-	-	-	-	29-5
Place of Manufacture									29-6
The "Software" Serial Number	-	-	-	-	-	-	-	-	29-6
Pure conjecture (Probably Incorrect)									29-6
A Little More History									29-6
 <b>Chapter 30: The Advanced Encryption Standard</b>	-	-	-	-	-	-	-	-	30-1
The Need for a New Encryption Standard									30-1
The New Encryption Standard	-	-	-	-	-	-	-	-	30-1
The Rounds									30-1
Copy Bits Logical (CPYBTL)	-	-	-	-	-	-	-	-	30-2
Generated RISC Code for MI-Version									30-3
Generated RISC Code for C-Version	-	-	-	-	-	-	-	-	30-3
Optimized RISC Code for C-Version									30-4

C-Version of AES	-	-	-	-	-	-	-	-	30-4
Defining an Encryption Round									30-5
Defining a Decryption Round	-	-	-	-	-	-	-	-	30-6
Key Expansion									30-6
The Encryption/Decryption Process	-	-	-	-	-	-	-	-	30-8
Operating Modes									30-8
Electronic Codebook Mode	-	-	-	-	-	-	-	-	30-9
Cipher Block Chaining Mode									30-9
The File Encryption Program, MIFAES	-	-	-	-	-	-	-	-	30-10
Encryption/Decryption Speed									30-11
Compiling and Binding the C-Programs	-	-	-	-	-	-	-	-	30-11
The AES C-Program									30-11
The AESTEST Testprogram	-	-	-	-	-	-	-	-	30-12
 <b>Chapter 31: The Object Information Repository</b>	-	-	-	-	-	-	-	-	 31-1
Extended Common Object Information									31-1
Work with Object Command (WRKOBJ)	-	-	-	-	-	-	-	-	31-1
The OIR (Object Information Repository)									31-2
Work with OIR (MIWRKOIR)	-	-	-	-	-	-	-	-	31-3
Must be System State									31-5
The Change Object Description API (QLICOBJD)	-	-	-	-	-	-	-	-	31-5
Fields in the OIR									31-5
The MISHWOIR Test Program	-	-	-	-	-	-	-	-	31-7
What is in the Additional OIR Records?									31-11
 <b>Chapter 32: Immune against Check Object Integrity</b>	-	-	-	-	-	-	-	-	 32-1
Checking Object Integrity									32-1
Hacking CHKOBJITG?	-	-	-	-	-	-	-	-	32-1
Using CHKOBJITG									32-1
Disowning an Object	-	-	-	-	-	-	-	-	32-2
The User Profile as a Machine Index									32-3
Defense in Depth (The Alter Log)	-	-	-	-	-	-	-	-	32-3
Format of Alter Log Entries									32-4
Clearing the Alter Log	-	-	-	-	-	-	-	-	32-5
How Did We Know the Address of the Alter Log?									32-5
 <b>Chapter 33: Analysis of SCV 7, Program Call</b>	-	-	-	-	-	-	-	-	 33-1
Program Calls									33-1
Adopted Authority	-	-	-	-	-	-	-	-	33-1
The Program State Attribute									33-1
Generated Code for CALLX Instruction	-	-	-	-	-	-	-	-	33-1
The SCV 7 Dispatch Code									33-2
The #AICAPGM Module	-	-	-	-	-	-	-	-	33-2
Check State Attribute									33-3
Finally Calling the Program	-	-	-	-	-	-	-	-	33-4
Entering the Called Program									33-4
The STDU RISC-Instruction	-	-	-	-	-	-	-	-	33-4
Returning from the Called Program									33-4
Format of the Invocation Stack Frame, ISF	-	-	-	-	-	-	-	-	33-5
 <b>Chapter 34: Anatomy of a User Profile</b>	-	-	-	-	-	-	-	-	 34-1
The Central Role of a User Profile									34-1
Getting User Profile for a Job	-	-	-	-	-	-	-	-	34-1
Object Specific Header									34-2
Privileged Instruction Bits in User Profile	-	-	-	-	-	-	-	-	34-2
Special Authority Bits in User Profile									34-2

Object Audit Level	-	-	-	-	-	-	-	-	34-3
User Audit Levels									34-3
A Dangerous Program	-	-	-	-	-	-	-	-	34-4
The Associated Space									34-4
Qualified Values	-	-	-	-	-	-	-	-	34-5
Various User Profile Flags									34-5
Language, Country, and Character Set IDs	-	-	-	-	-	-	-	-	34-5
User Options									34-6
Objects Owned by User Profile	-	-	-	-	-	-	-	-	34-6
 <b>Chapter 38: Source and Debug Information</b>									
ILE Programs and Modules									38-1
OIR-Information for Modules	-	-	-	-	-	-	-	-	38-1
Change Licensed Object Description, QLICOBJD									38-2
Anatomy of a Module	-	-	-	-	-	-	-	-	38-2
The HLL Symbol Table									38-3
View Descriptors	-	-	-	-	-	-	-	-	38-3
Descriptor Types									38-5
No Debug Source	-	-	-	-	-	-	-	-	38-5
Why Make the Symbol Table Inaccessible?									38-5
Changing Source/Debug Information, MICHGMOD					-	-	-	-	38-5
The Bound Program									38-8
The BNAS Information	-	-	-	-	-	-	-	-	38-8
Debug Source Information									38-11
 <b>Appendix A: MI-Instructions Quick Reference</b>									
MI-Instruction Quick Reference									A-1
Computational and Branching Instructions	-	-	-	-	-	-	-	-	A-1
Pointer/Resolution Instructions									A-6
Space Management Instructions	-	-	-	-	-	-	-	-	A-7
Independent Index Instructions									A-7
Authorization Instructions	-	-	-	-	-	-	-	-	A-7
Program and Invocation Instructions									A-8
Exception Management Instructions	-	-	-	-	-	-	-	-	A-9
Queue Management Instructions									A-9
Object Lock Instructions	-	-	-	-	-	-	-	-	A-10
Context Management Instructions									A-10
Heap Management Instructions	-	-	-	-	-	-	-	-	A-10
Resource Management Instructions									A-11
MI Support Functions Instructions	-	-	-	-	-	-	-	-	A-11
Date/Time/Timestamp Instructions									A-11
 <b>Appendix B: Object Type/Subtypes</b>									
Object Type/Subtypes									B-1
Object List	-	-	-	-	-	-	-	-	B-1
 <b>Appendix C: PowerPC Instruction Set Quick Reference</b>									
Instructions Sorted by Mnemonic									C-1
 <b>Appendix D: SEPT Entries in the User Domain</b>									
System Entry Point Table Entries									D-1
Entries Sorted by Number	-	-	-	-	-	-	-	-	D-1
Entries Sorted by Name									D-13
 <b>Appendix E: SCV 10 Function Numbers</b>									
Supervisor Call (Vectored) Functions									E-1

Appendix F: <b>Open and I/O Feedback Areas</b>	-	-	-	-	-	F-1
Open Feedback Area						F-1
I/O Feedback Area	-	-	-	-	-	F-6
Common I/O Feedback Area						F-6
I/O Feedback Area for ICF and Display Files	-	-	-	-	-	F-8
I/O Feedback Area for Printer Files						F-10
I/O Feedback Area for Database Files	-	-	-	-	-	F-10

# Chapter 0

## Introduction

### ***Why Program at the Machine-Level?***

Leaving aside the precise definition of Machine-Level for a bit, a good reason is that it is just plain ol' *fun*. It certainly is different from cranking out RPG-programs. Then there are all the standard reasons that have to do with wringing more performance out of your system with the usual caveats about only optimizing what is worth optimizing. There are many APIs on the AS/400 that are just machine-level instructions in disguise, and at times bad ones at that. When you are using such an API, you are doing machine-level programming, but often without understanding precisely what is going on and often forced into a clumsy and tedious format required by the high-level language you are using. If you know how to program at the machine-level, you will find it easier to use APIs from your high-level language, and will use them with more confidence. Finally, there are things that you just can't do in any reasonable, other way without going to the machine-level.

### ***What is the Machine-Level?***

One definition is that it simply is the lowest level at which you can work as a programmer. This level constitutes an *abstract* machine, and you program to that abstraction. There are always levels below that you ordinarily don't care about, and in fact often don't want even to *know* about. The AS/400 is somewhat unusual because there are *two* machine-levels that play a role in practice. The upper one is called the MI-level and the lower one is the CISC/RISC platform. MI, or the *Machine Interface*, is in a very real sense what makes your machine an AS/400 rather than just a souped-up PowerPC. This book will show you how to program to the MI-level. We shall also examine the RISC platform in detail so you will understand something of what goes on "under the covers".

### ***Above and Below the MI***

The operating system that controls your machine has two parts. Over time some misleading nomenclature has been used. You have probably heard about "horizontal" and "vertical" micro-code. These names have fallen out of favor, mainly for legal reasons. Since "micro-code" is considered part of the hardware you can *own* micro-code. IBM doesn't want you to own the operating system, so requires you to *license* it instead, hence had to change the names. Today, names like OS/400 and SLIC (System Licensed Internal Code) are used. Basically, OS/400 is programmed to the MI-level and SLIC is programmed to the PowerPC RISC platform. This is often expressed by saying that OS/400 (and your applications) are *above* the MI and SLIC is *below* the MI, hence justifying talking about the machine *interface*.

### ***Old MI and New MI***

Just as the AS/400 hardware has evolved, MI has too. MI was designed to be extensible and new operations and functionality have been added over time as needed. We are at a point now where one can talk about the "old" or classic MI supporting the "old programming model", OPM, and the "new" MI supporting the ILE programming model with its emphasis on C-style programs. Today's RISC-based AS/400 only support the ILE programming model, but a special module in SLIC takes care of transforming OPM program objects into ILE modules bound into an ILE program. The module that does that has been called the "Magic" module. There is this notion that there is some magic involved in MI-programming. I don't like magic. There is a famous quote from the Science Fiction master Arthur C. Clarke that "any sufficiently advanced technology is indistinguishable from magic". One purpose of this book is to dispel some of the magic by seeking an actual understanding of what is happening.

### ***Is MI Hard and Arcane?***

It is a common misconception that machine-level programming is *hard*. There may be some truth to that at the RISC level, but that certainly is not so at the MI-level. MI is a very expressive language with powerful data structuring facilities that makes programming easy and straightforward. I can still remember my very

first MI-program (some time back in 1989). I converted a 2000-line COBOL program into MI in less than a week and it ran the first time dropping the time it took to generate a 5250 datastream from 0.337 seconds to 0.017 seconds for a speed-up factor of 20. If I can do it, so can you.

## ***What About All Those MI-Instructions?***

Analysis of several large production-type MI-programs containing thousands of instructions show that only 10 instructions comprise almost 80% of all instructions used:

<b>Mnemonic</b>	<b>Freq</b>	<b>% of Total</b>	<b>Instruction Description</b>
CPYBLA	19.89 %	19.89 %	Copy Bytes Left Adjusted
CPYNV	10.86 %	30.75 %	Copy Numeric Value
B	10.03 %	40.78 %	Branch
CMPNV	9.79 %	50.57 %	Compare Numeric value
ADDN	7.95 %	58.52 %	Add Numeric
CMPBLA	6.53 %	65.05 %	Compare Bytes Left Adjusted
CPYBLAP	4.27 %	69.32 %	Copy Bytes Left Adjusted with Pad
CALLX	3.80 %	73.12 %	Call External (program)
SUBN	3.62 %	76.74 %	Subtract Numeric
CALLI	2.67 %	79.41 %	Call Internal (subroutine)

Fundamental operations include copying characters (**CPYBLA** and **CPYBLAP**), copying numbers (**CPYNV**), branching (**B**), and comparisons (**CMPBLA** for characters and **CMPNV** for numbers). These alone make up more than 61% of all instructions coded. The moral of this exercise was to show that a lot could be accomplished with a little, so you should be able to quickly become productive. Ovid said “Add little to little and there will be a big pile”, same thing here.

## ***Small Programs or Large Programs***

Some people advocate only writing very small programs in MI. Useful programs can be as short as a single instruction (with an implied return instruction). The argument is that maintaining MI programs is hard. This is actually not the case, rather, well-written MI-programs are easy to maintain (as are most *well-written* programs in any language). What *was* true, was that finding people with the skills needed was hard. That is another one of the reasons for this very book. When you have *worked* your way through the book, you will find that acquiring MI-skills was not all that hard.

I know of whole applications written solely in MI, comprising tens of thousands of lines of source code. Experience shows that these applications are not any harder to maintain than applications written in other languages. Because of the interoperability of programs on the AS/400 (one of the delivered promises of ILE) it probably would make good sense to write pieces of the application in languages best suited for that particular piece, with MI taking its place among the others on an equal footing doing what *it* does best.

## ***Why is MI so Secret?***

As we all know, IBM has not been very helpful in supplying information about MI-programming. The hoped for support, expressed in NEWS 3X/400 September 1989, that “You may decide that you would like IBM to let the S/38 and AS/400 “be all that they can be” by openly supporting MI” did not come to pass. There have been a handful of articles in the various AS/400 publications, and only recently has there been a mailing list (MI400@MIDRANGE.COM) to cater for the curious-minded AS/400 programmers. The IBM public documentation for the MI language is very sparse. The MI Functional Reference Manual, which describes each (of a subset, only) MI instruction doesn’t even have one single MI example within it’s many pages. The System API Reference manual has a tiny chapter, dedicated to the MI language syntax, but all of this information not really enough to be a working set of reference materials for programming in MI. Maybe this book will help you to make the AS/400 “be all that it can be”?

# Chapter 1

## Getting Your Own MI-Compiler

### MI-Compilers

If you want to program in MI, you of course will need a compiler. While you could purchase an MI-compiler for the S/38 from IBM (the “PRPQ”), IBM claimed that *no* MI-compiler for the AS/400 was available. Several people discovered early on that contrary to this claim, every AS/400 was, in fact, shipped with an MI-compiler built in. What is missing is a convenient way to invoke the compiler, i.e. there is no **CRTMI PGM** command. Some people would you sell a simple front-end program as “An MI-compiler” for several thousand dollars. Maybe to stop that practice (or for reasons unknown), IBM finally documented an API named QPRCRTPG (Create Program) to invoke the compiler. You can find this information (at least until IBM pulls it again) at this web-site:

[“http://as400bks.rochester.ibm.com/cgi-bin/bookmgr/bookmgr.cmd/BOOKS/QB3AVC00/7.0”](http://as400bks.rochester.ibm.com/cgi-bin/bookmgr/bookmgr.cmd/BOOKS/QB3AVC00/7.0).

The method described at the above URL is needlessly complex and even wrong if you follow the instructions to the letter (hint: one of the CL-programs blows up because the source presented to it exceeds the 2000-character limit imposed on a variable in CL) and as such we shall not duplicate that material here. Instead we shall write an RPG-program that will invoke the API. If you do not have an RPG compiler, you can download a ready-made savefile with the resulting compiler front-end from our website at this URL: <http://iSeries400.org/micomp.exe>. You will learn more from trying to build one yourself following the steps given in this book.

The Create Program (**QPRCRTPG**) API converts the symbolic representation (read: simple source text) of a machine interface (MI) program into an OPM program object. This symbolic representation is known as the intermediate representation of a program. The **QPRCRTPG** API creates a program object that resides in the \*USER domain and runs in the \*USER state. In later chapters we shall examine the domain/state concepts in detail.

### Create Program (**QPRCRTPG**) API

The **QPRCRTPG** API is documented in the books for V3R2 and in the System API manual for later releases. Here is an on-line reference <http://publib.boulder.ibm.com/pubs/html/as400/online/v3r2eng.htm>. Below I excerpt here the relevant information for your convenience (who knows for how long the above link is still good?). Much of the detailed description (shown in a smaller font) may not make much sense to your at this stage, so you should just consider it to be reference material, skim it, and continue with confidence.

What makes the API a little tricky to use is that you do not pass it the name of a member in a source file as you would ordinarily do, but instead, you pass the API a large character array containing the source. For this reason, a *front-end* is usually required to read the source records into an array. Here is first the parameter list for the API:

1	Program source statements	In	Char(*)
2	Length of program source statements	In	Binary(4)
3	Qualified program name	In	Char(20)
4	Program text	In	Char(50)
5	Qualified source file name	In	Char(20)
6	Source file member information	In	Char(10)
7	Source file last changed date and time	In	Char(13)
8	Qualified printer file name	In	Char(20)
9	Starting page number	In	Binary(4)
10	Public authority	In	Char(10)
11	Option template	In	Char(*)
12	Number of option template entries	In	Binary(4)
13	Error code (optional)	I/O	Char(*)



## Detailed Explanation of Each Parameter (Boring)

**Program source statements** INPUT; CHAR(\*)

A string containing the MI-source statements of the program to be processed by the API. We must read the source member into this area. The reason for this is that MI was often the intermediate output from a HLL compiler and therefore never was placed in a source member.

**Length of program source statements** INPUT; BINARY(4)

The size, in characters, of the source statement string.

**Qualified program name** INPUT; CHAR(20)

The name and library of the program to be created or replaced. The first 10 characters contain the program name, and the second 10 characters contain the name of the library where the program is located. \*CURLIB may be used for the library name.

**Program text** INPUT; CHAR(50)

Text that briefly describes the program.

**Qualified source file name** INPUT; CHAR(20)

The name and library containing the source program. The first 10 characters contain the source file name, and the second 10 characters contain the name of the library where the file is located. This places the value in the program object's service description. \*NONE may be used for the source file name, in which case no source file information is placed in the description. The source file library must be explicitly given if used. Note, that the API does not read the source from this member.

**Source file member information** INPUT; CHAR(10)

The file member containing the source program. This places the value in the program object's service description. This value must be blanks if you specify \*NONE as the **source file name**.

**Source file last changed date and time information** INPUT; CHAR(13)

The date and time the member of the source file was last changed in the format CYYMMDDHHMMSS format, where: C=0 indicates years 19xx and 1 indicates years 20xx. This places the value in the program object's service description. This value must be blank if you specify \*NONE for the source file name parameter. Note, that the API uses the value you supply, rather than the actual date when the source was changed.

**Qualified printer file name** INPUT; CHAR(20)

The name and library containing the printer file used to generate listings. The first 10 characters contain the printer file name, and the second 10 characters contain the name of the library where the file is located. You can use \*LIBL and \*CURLIB for the library as appropriate. This value is ignored if you specify \*NOLIST for the generate listing option.

**Starting page number** INPUT; BINARY(4)

The first page number to be used on listings. This value is ignored if you specify \*NOLIST for the generate listing option.

**Public authority** INPUT; CHAR(10)

The authority you give the users who do not have specific private authorities to the object. The values allowed are the usual: \*CHANGE, \*ALL, \*USE, \*EXCLUDE, or the name of an authorization list.

**Option template** INPUT; CHAR(\*)

This is an array of options with from 0 to 16 values. Each entry contains a CHAR(11) value. If you specify an option more than once, the system only uses the first one. If you omit an option, the system uses a default value (underlined).

**Create program object** Creates a program object:

\*GEN

Generates a program in the appropriate library.

\*NOGEN

No program is generated. The syntax of the program is checked, and if the generate listing option is \*LIST, a listing is produced.

**Replace program** Replaces the existing program if a program by the same name already exists in the specified library:

\*NOREPLACE

Does not replace an existing program by the same name in the specified library.

\*REPLACE

Replaces the existing program by moving it to the QRPLOBJ library.

**Generate listing** Generates an output listing:

\*NOLIST

Does not generate a listing.

\*LIST

Generates a listing. You must specify the following parameters: Printer file name and library and Starting page number

**Create cross-reference listing** Whether the listing is to contain a cross-reference list of variable references:

- \*NOXREF Does not create cross-reference listing.
- \*XREF Creates a cross-reference listing of references to variables.

**Create summary listing** Whether the listing is to contain a list of program attributes:

- \*NQATR Does not create a summary listing section.
- \*ATR Creates a summary listing section.

**User profile** The values allowed are:

- \*USER The user profile of the user running the program is used as a source of authority.
- \*ADOPT The object authority of both the program's owner and user are used.
- \*OWNER The system uses the user profile of the owner of the program as a source of authority when this program runs. Programs called by this program adopt this authority.

**Use adopted authority** Whether the system uses the program-adopted authority from the calling programs as a source of authority when this program is running. The user must be authorized to create programs with adopted authority for the \*ADPAUT option to take effect:

- \*ADPAUT The system uses program-adopted authority from the calling program.
- \*NOADPAUT The system does not use program-adopted authority from the calling program.

**Note:** Authorization to create programs which can adopt authority is controlled by the QUSEADPAUT system value.

**Constrain arrays** The values allowed are:

- \*SUBSCR Constrains arrays. This requests additional run-time checks to ensure that references to array elements are not outside the bounds of the declare statement.
- \*NOSUBSCR Does not constrain arrays. The results of references to array elements outside the bounds of the declare statement are not defined.
- \*UNCON Allows fully unconstrained arrays. This ensures that references to array elements outside the bounds of the declare statement act as if the array element actually exists.

**Constrain strings** The values allowed are:

- \*SUBSTR Constrains strings. This requests additional run-time checks to ensure that references to character strings are not outside the bounds of the declare statement. This option causes the resulting program to run slower.
- \*NOSUBSTR Does not constrain strings. The results of substring references outside the bounds of the declare statement are not defined.

**Initialize static storage** **Static storage** is allocated the first time a program is called. It remains allocated until explicitly deallocated:

- \*CLRPSSA Initializes static storage.
- \*NOCLRPSSA Does not initialize the PSSA.

**Initialize automatic storage** **Automatic storage** is allocated each time a program runs and automatically deallocated when no longer needed:

- \*CLRPASA Initializes automatic storage.
- \*NOCLRPASA Does not initialize the PASA.

**Ignore decimal data errors** Whether errors found in decimal data result in exceptions:

- \*NOIGNDEC Does not ignore decimal data errors.
- \*IGNDEC Ignores data decimal errors.

**Ignore binary data size errors** The values allowed are:

- \*NOIGNBIN The system handles binary data size errors normally.
- \*IGNBIN The system ignores binary data size errors. This is used when an overflow or underflow occurs when an MI instruction has a receiver that is a binary field.

**Support coincident operands** The system overlaps coincident operands between the source and receiver operands in one or more program instructions. **Coincident operands** are operands that overlap physically, in storage:

- \*NOOVERLAP Does not support coincident operands. If you specify \*NOOVERLAP, you guarantee that coincident operand overlap will not occur while running the instruction.
- \*OVERLAP Supports coincident operands. If you specify \*OVERLAP, the operands on an MI instruction may overlap.

**Allow duplicate declares** The values allowed are:

- \*NODUP This does not allow a program object to be declared more than once.
- \*DUP This allows a program object to be declared more than once.

**Optimize** The values allowed are:

- \*OPT This optimizes the program producing the smallest and best running program.
- \*NOOPT This does not optimize the program beyond the normal level code optimization.

#### Number of option template entries INPUT; BINARY(4)

The number of option template entries. The value must be between 0 and 16.

**Error code** I/O; CHAR(\*)

The structure in which to return error information. If BINARY(4) with value 0 is specified, no detailed error information is returned.

## The MI-Compiler Front-End

We must first decide which language to use for our compiler front-end. Every AS/400 has a CL-compiler, but the limitation of 2000 characters for the size of the variable to hold the MI source string is severe. Most AS/400 shops have an RPG-compiler, but even RPG has limitations. Other languages are not used broadly enough to be viable for my purpose, so I decided to write the front-end in MI! Of course, I still have to compile the front-end, but since the front-end code is small (about 200 lines) I can embed the code in an RPG array and have a simple RPG-program call the **QPRCRTPG** API to generate the MI front-end program.

```
/*=====
 * This program creates MI compiler front-end CRTMI PGM in *CURLIB=
 * Source statements for the MI compiler are found in array MI. =
 *=====
E          DS          MI          1 208 80
I
I          DS          CRTMI PGM *CURLIB
I          DS          *NONE
I
I          DS          B 1 40#SRCLN
I          DS          5 24 #PGMLB
I          DS          25 74 #TEXT
I          DS          75 94 #SRCFL
I          DS          95 104 #MBR
I          DS          105 117 #CHGDT
I          DS          105 105 #CENT
I          DS          106 107 #YY
I          DS          108 111 #MMDD
I          DS          112 117 #HMS
I          DS          118 137 #PRTFL
I          DS          B 138 1410#STRPG
I          DS          142 151 #AUT
I          DS          152 327 #OP
I          DS          B 328 3310#NOOPT
C          CALL 'QPRCRTPG'
C          PARM MI
C          PARM 16640 #SRCLN
C          PARM #PGMLB
C          PARM 'MI Comp' #TEXT
C          PARM #SRCFL
C          PARM #MBR
C          PARM #CHGDT
C          PARM ' ' #PRTFL
C          PARM 0 #STRPG
C          PARM '*USE' #AUT
C          PARM '*REPLACE' #OP
C          PARM 1 #NOOPT
C          MOVE *ON *INLR
```

You can easily recognize the parameters to the **QPRCRTPG** API. All we have to do now is to populate the array MI with the appropriate code (as shown below), compile the RPG program and run it. The result is the **CRTMI PGM** program object. Now, you should try this right away.

Below are the contents of the MI array. The curious **\*/** in the first line has a matching **/\*** at the beginning of the RPG-program. Comments in MI are free-form text (can be spread over several lines) starting with **/\*** and ending with **\*/**. That makes the entire code part of the RPG-program an MI-comment. In fact, the program is at the same time a valid RPG-program *and* a valid MI-program. You can give it to either compiler and it will compile and run. Why would I do a thing like this? Maintenance. I can enhance the front-end and make it a real pre-compiler supported added functionality. I have, in fact, done that already with the **%INCLUDE** facility. I can then test the result as I would test any other MI-program.

Don't worry yet about how the code in the array works. We shall get to all that in due time. For now, just look at it as an example of a non-trivial MI-program. This is not a toy program

Here is then, finally, the RPG-array:

```
**
DCL SPCPTR .MBR PARM;
DCL SPCPTR .FIL PARM;
```

```

DCL SPCPTR .DET PARM;
DCL OL *ENTRY (.MBR, .FIL, .DET) PARM EXT MIN(1);
DCL DD MBR CHAR(10) BAS(.MBR);
DCL DD FIL CHAR(10) BAS(.FIL);
DCL DD DET CHAR(10) BAS(.DET);

DCL SPC PCO BASPCO;
DCL SPCPTR .PCO DIR;

DCL SPC SEPT BAS(.PCO);
DCL SPCPTR .SEPT(2000) DIR;

DCL SPCPTR .UFCB INIT(UFCB);
DCL DD UFCB CHAR(214) BDRY(16);
DCL SPCPTR .ODP DEF(UFCB) POS( 1);
DCL SPCPTR .INBUF DEF(UFCB) POS( 17);
DCL SPCPTR .OUTBUF DEF(UFCB) POS( 33);
DCL SPCPTR .OPEN-FEEDBACK DEF(UFCB) POS( 49);
DCL SPCPTR .IO-FEEDBACK DEF(UFCB) POS( 65);
DCL SPCPTR .NEXT-UFCB DEF(UFCB) POS( 81);

DCL DD * CHAR(32) DEF(UFCB) POS( 97);
DCL DD FILE CHAR(10) DEF(UFCB) POS(129) INIT("QMI SRC");
DCL DD LIB-ID BIN ( 2) DEF(UFCB) POS(139) INIT(-75);
DCL DD LIBRARY CHAR(10) DEF(UFCB) POS(141) INIT("*LIBL");
DCL DD MBR-ID BIN ( 2) DEF(UFCB) POS(151) INIT( 73);
DCL DD MEMBER CHAR(10) DEF(UFCB) POS(153);

DCL DD ODP-DEVICE-NAME CHAR(10) DEF(UFCB) POS(163);
DCL DD ODP-DEVICE-INDEX BIN ( 2) DEF(UFCB) POS(173);

DCL DD FLAGS-PERM-80 CHAR( 1) DEF(UFCB) POS(175) INIT(X' 80');
DCL DD FLAGS-GET-20 CHAR( 1) DEF(UFCB) POS(176) INIT(X' 20');
DCL DD REL-VERSION CHAR( 4) DEF(UFCB) POS(177) INIT("0100");
DCL DD INVOC-MARK-COUNT BIN ( 4) DEF(UFCB) POS(181);
DCL DD MORE-FLAGS CHAR( 1) DEF(UFCB) POS(185) INIT(X' 00');
DCL DD * CHAR(23) DEF(UFCB) POS(186);

DCL DD RECORD-PARAM BIN ( 2) DEF(UFCB) POS(209) INIT(1);
DCL DD RECORD-LENGTH BIN ( 2) DEF(UFCB) POS(211) INIT(92);

DCL DD NO-MORE-PARAMS BIN ( 2) DEF(UFCB) POS(213) INIT(32767);

DCL SPC ODP BAS(.ODP);
DCL DD * CHAR(16) DIR;
DCL DD DEV-OFFSET BIN ( 4) DIR;

DCL SPCPTR .DMDEV;
DCL SPC DMDEV BAS(.DMDEV);
DCL DD MAX-DEVICE BIN ( 2) DIR;
DCL DD NBR-DEVICES BIN ( 2) DIR;
DCL DD DEVICE-NAME CHAR(10) DIR;
DCL DD WORKAREA-OFFSET BIN ( 4) DIR;
DCL DD WORKAREA-LENGTH BIN ( 4) DIR;
DCL DD LUD-PTR-INDEX BIN ( 2) DIR;
DCL DD DM-GET BIN ( 2) DIR;

DCL SPCPTR .GETOPT INIT(GETOPT);
DCL DD GETOPT CHAR(4);
DCL DD GET-OPTION-BYTE CHAR(1) DEF(GETOPT) POS(1) INIT(X' 03');
DCL DD GET-SHARE-BYTE CHAR(1) DEF(GETOPT) POS(2) INIT(X' 00');
DCL DD GET-DATA-BYTE CHAR(1) DEF(GETOPT) POS(3) INIT(X' 00');
DCL DD GET-DEVICE-BYTE CHAR(1) DEF(GETOPT) POS(4) INIT(X' 01');

DCL SPCPTR .NULL;
DCL OL GET (.UFCB, .GETOPT, .NULL);
DCL OL OPEN (.UFCB);
DCL OL CLOSE(.UFCB);

DCL SPC INBUF BAS(.INBUF);
DCL DD INBUF-DATE CHAR(12) DEF(INBUF) POS( 1);
DCL DD INBUF-LINE CHAR(80) DEF(INBUF) POS(13);
DCL DD INBUF-KEYWORD CHAR( 9) DEF(INBUF-LINE) POS( 1);
DCL DD INBUF-NEWMBR CHAR(10) DEF(INBUF-LINE) POS(10);

DCL SPCPTR .SOURCE;
DCL DD LINE(10000) CHAR(80) AUTO;
DCL DD LINE-NBR BIN(4);
DCL DD READ-NBR BIN(4);
DCL DD SAVE-NBR BIN(4);
DCL DD SKIP-NBR BIN(4);

```

```

DCL DD INCL-NBR BIN(2);

DCL SPCPTR .SIZE INIT(SIZE);
DCL DD      SIZE BIN(4);

DCL SPCPTR .PGM INIT(PGM);
DCL DD      PGM CHAR(20);
DCL DD PGM-NAME CHAR(10) DEF(PGM) POS( 1);
DCL DD PGM-LIB  CHAR(10) DEF(PGM) POS(11) INIT(" *CURLIB");

DCL SPCPTR .PGM-TEXT INIT(PGM-TEXT);
DCL DD      PGM-TEXT CHAR(50) INIT(" ");

DCL SPCPTR .PGM-SRCF INIT(PGM-SRCF);
DCL DD      PGM-SRCF CHAR(20) INIT(" *NONE");

DCL SPCPTR .PGM-SRCM INIT(PGM-SRCM);
DCL DD      PGM-SRCM CHAR(10) INIT(" ");

DCL SPCPTR .PGM-SRCD INIT(PGM-SRCD);
DCL DD      PGM-SRCD CHAR(13) INIT(" ");

DCL SPCPTR .PRTF-NAME INIT(PRTF-NAME);
DCL DD      PRTF-NAME CHAR(20);
DCL DD PRTF-FILE CHAR(10) DEF(PRTF-NAME) POS( 1) INIT("OSYSVRT ");
DCL DD PRTF-LIB  CHAR(10) DEF(PRTF-NAME) POS(11) INIT(" *LIBL ");

DCL SPCPTR .PRT-STRPAG INIT(PRT-STRPAG);
DCL DD      PRT-STRPAG BIN(4) INIT(1);

DCL SPCPTR .PGM-PUBAUT INIT(PGM-PUBAUT);
DCL DD      PGM-PUBAUT CHAR(10) INIT(" *ALL");

DCL SPCPTR .PGM-OPTS INIT(PGM-OPTS);
DCL DD      PGM-OPTS(16) CHAR(11) INIT(" *REPLACE ", " *NOADPAUT ",
                                     " *NOCLRPSSA ", " *NOCLRPASA ", " *SUBSCR ",
                                     " *LIST ", " *ATR ", " *XREF ");

DCL SPCPTR .NBR-OPTS INIT(NBR-OPTS);
DCL DD      NBR-OPTS BIN(4);

DCL OL QPRCRTPG (.SOURCE, .SIZE, .PGM, .PGM-TEXT, .PGM-SRCF,
                 .PGM-SRCM, .PGM-SRCD, .PRTF-NAME, .PRT-STRPAG,
                 .PGM-PUBAUT, .PGM-OPTS, .NBR-OPTS) ARG;

DCL SYSPTR .QPRCRTPG INIT("QPRCRTPG", CTX("QSYS"), TYPE(PGM));

DCL DD NBR-PARMS BIN(2);
DCL EXCM * EXCID(H' 5001') BP(EOF) IMD;

DCL DD START CHAR(80);
DCL DD * CHAR(12) DEF(START) POS( 1) INIT("/ * INCLUDE: ");
DCL DD NEWMBR CHAR(10) DEF(START) POS(13);
DCL DD * CHAR(58) DEF(START) POS(23) INIT(" */");

DCL DD STOP CHAR(80);
DCL DD * CHAR(80) DEF(STOP) POS(1) INIT("/ * END INCLUDE */");

/*****/

ENTRY * (*ENTRY) EXT;
CPYNV LINE-NBR, 1;
CPYNV INCL-NBR, 0;
CPYNV SKIP-NBR, 0;

CPYBWP .NULL, *;
CPYNV NBR-OPTS, 6; /* YES: *LIST; NO: *ATR, *XREF */
STPLLEN NBR-PARMS;
CMPNV(B) NBR-PARMS, 3/NEQ(PREPARE-FILE);
CMPBLA(B) DET, <10|*DETAIL >/EQ(YES-DETAIL);
CMPBLA(B) DET, <10|*NOLIST >/EQ(NO-LIST);
B PREPARE-FILE;
YES-DETAIL: CPYNV(B) NBR-OPTS, 8/NNAN(PREPARE-FILE);
NO-LIST: CPYNV(B) NBR-OPTS, 5/NNAN(PREPARE-FILE);

PREPARE-FILE:
CPYBLAP FILE, "QMISRC", " ";
CMPNV(B) NBR-PARMS, 1 /EQ(SET-MEMBER);
CPYBLA FILE, FILE;
SET-MEMBER:
CPYBLA MEMBER, MBR;

```

The curious statement “CPYBLAP LINE(LINE-NBR), <23|/\*/\*/\*/\*\*/; PEND;;;>, “ ”;” catches strings and comments that are not closed. Without this statement, the translator is prone to crashing (“dumping”) if end-of-file is encountered and a string or comment is still open.

```

                                Create Command (CRTCMD)
Type choices, press Enter.
Command . . . . . > CRTMI PGM      Name
Library . . . . . > LSVALGAARD    Name, *CURLIB
Program to process command . . . > CRTMI PGM    Name, *REXX
Library . . . . . > *LIBL         Name, *LIBL, *CURLIB
Source file . . . . . > QCMDSRC     Name
Library . . . . . > LSVALGAARD    Name, *LIBL, *CURLIB
Source member . . . . . > CRTMI PGM  Name, *CMD
Threadsafe . . . . . > *NO         *YES, *NO, *COND

```

You can either run it from the command line:

```
====> CRTMI PGM PGM(yourpgm) FILE(QMI SRC) LIST(*LIST)
```

or (if the defaults are to your liking):

```
====> CRTMI PGM yourpgm
```

You can also prompt the command:

Create MI program (CRTMI PGM)		
Type choices, press Enter.		
Program Source Member . . . . .	<u>yourpgm</u>	Character value
in Source File . . . . .	<u>QMI SRC</u>	Character value
LIST option . . . . .	<u>*LIST</u>	Character value

Note, that the source file is by default **QMI SRC** on the library list. The following *options* are selected automatically in the front-end:

```
*REPLACE *NOADPAUT *NOCLRPSSA *NOCLRPASA *SUBSCR *LIST *ATR *XREF
```

The **\*LIST** option on the command cuts the last two options from the above set, meaning, in effect **\*LIST** only. **\*NOLIST** cuts the last three, and **\*DETAIL** cuts none.

At this point I suggest that you copy the RPG-source of **CRTMI PGM** to a member with a different name in **QMI SRC** (create the 92-character source file first, if needed), e.g. **QMI SRC/TEST**, then try to compile **TEST**:

```
====> CRTMI PGM test
```

A listing can be found in the spoolfile **QSYSPRT**. Before continuing, make sure that all this works and that you understand the process (even if the code is still voodoo or ‘magic’).

## Hello World

It is customary in programming language books to start your first programming lesson by showing how to write a program that displays the text “Hello World” in that particular language. Some languages ask you to marvel over the fact that this program can be written as a single statement, in which case, of course, the exercise is vacuous, because you haven’t learnt anything by writing **display “Hello World”**. Here is what an MI-version might look like:

```
CPYBLA MSG-TEXT, "Hello World", " ";
CALLI SHOW-MESSAGE, *, . SHOW-MESSAGE;
RTX *;
%I INCLUDE SHOWMSG
```

The first line copies (**CPY**) the bytes (**B**) left-aligned (**LA**) from the 11-bytes long literal “Hello World” to the variable **MSG-TEXT**, then pads (**P**) the result with blanks (“ ”) until the end of the receiver. Note, that instructions are generally executing from right-to-left, so that “CPYBLA A, B” copies B to A. This is the opposite of RPG’s and COBOL’s “MOVE A to B” that work left-to-right. This may take some time to get used to, but is similar to mathematical expressions, e.g.  $A = B$ , means assign B to A. Or  $A = \text{sqrt}(B)$ , which means: take B, extract its square-root, then assign the result to A.

The second line calls (**CALL**) the internal (**I**) subroutine **SHOW-MESSAGE**. Which presumably is going to show the message (possibly on the display device). Finally, the program returns (**RT**) to its external caller (**X**). Commas separate operands and a semi-colon (;) terminates each instruction. Instructions are conventionally written one to a line, but they don’t need to be. MI is completely free-form. Except for literal strings, everything must be in UPPER CASE. This is one of the restrictions we will relax in a later chapter when we introduce more pre-processing facilities in our MI front-end.

We haven’t said anything about the fourth line (**%I INCLUDE SHOWMSG**). We are hiding something messy here. The display the text, we are sending a message to a job using the **QMHSNDM** API. This API is *not* at the MI-level, but is a rather high-level system function. There is nothing that says that we have to stay at the

machine-level all the time. In fact, if there is a higher-level function that does the job using a reasonable amount of resources, by all means use it. The problem with **QMHSNDM** is that it takes *ten* parameters specifying all kinds of complicated details that we really don't want to know about. There are two main approaches to this problem, the first is to make a separate, external program that you simply call with the text to show, the second is to have an internal subroutine that you just call, after having loaded a common variable (**MSG-TEXT**) with the text to display. The declaration, variables, and code for the subroutine are held in a separate source member (QMI SRC/SHOWMSG) and are simply included into your MI-program. This is one of the main reasons why it was handy to provide an include facility in our pre-processing MI front-end.

One thing of note here, is that as stated before, MI is completely free form, notice how we included the SHOWMSG source file at the *bottom* of the MI HELLO source file. In languages such as RPG, all variable definitions must be before the first calculation line so that, in essence, they are defined before they are first referenced. In MI there is no such restriction on the placement of variable definitions.

Below is the code to store in the QMI SRC/SHOWMSG source member; we'll discuss how it works in a later chapter, but I guess that you can see that it basically defines and initializes the parameters to **QMHSNDM** and then calls the API with those values:

```
/* SHOW A MESSAGE */

DCL SPCPTR .MSG-ID    INIT(MSG-ID);
DCL DD      MSG-ID    CHAR(7) INIT(" ");

DCL SPCPTR .MSG-FILE  INIT(MSG-FILE);
DCL DD      MSG-FILE  CHAR(20) INIT(" ");

DCL SPCPTR .MSG-TEXT  INIT(MSG-TEXT);
DCL DD      MSG-TEXT CHAR(70);

DCL SPCPTR .MSG-SIZE  INIT(MSG-SIZE);
DCL DD      MSG-SIZE  BIN( 4)  INIT(70);

DCL SPCPTR .MSG-TYPE  INIT(MSG-TYPE);
DCL DD      MSG-TYPE  CHAR(10) INIT(" *INFO");

DCL SPCPTR .MSG-QS    INIT(MSG-QS);
DCL DD      MSG-QS    CHAR(20) INIT(" *REQUESTER");

DCL SPCPTR .MSG-QSN   INIT(MSG-QSN);
DCL DD      MSG-QSN   BIN( 4)  INIT(1);

DCL SPCPTR .REPLY-Q   INIT(REPLY-Q);
DCL DD      REPLY-Q   CHAR(20) INIT(" ");

DCL SPCPTR .MSG-KEY   INIT(MSG-KEY);
DCL DD      MSG-KEY   CHAR( 4);

DCL SPCPTR .ERR-CODE  INIT(ERR-CODE);
DCL DD      ERR-CODE  BIN( 4)  INIT(0);

DCL OL QMHSNDM (.MSG-ID, .MSG-FILE, .MSG-TEXT, .MSG-SIZE,
               .MSG-TYPE, .MSG-QS, .MSG-QSN, .REPLY-Q,
               .MSG-KEY, .ERR-CODE) ARG;

DCL SYSPTR .SEPT(6440) BAS(SEPT-POINTER);
DCL SPC PROCESS-COMMUNICATION-OBJECT BASPCO;
DCL SPCPTR SEPT-POINTER DIR;

DCL INSPTR .SHOW-MESSAGE;
ENTRY      SHOW-MESSAGE INT;
CALLX      .SEPT(4268), QMHSNDM, *; /* SEND MSG TO MSGQ */
B          .SHOW-MESSAGE;
```

Enter, compile, and run the MI HELLO program. You should see a result similar to this:

Display Messages			
Queue . . . . .	LSVALGAARD	Program . . . . .	*DSPMSG
Library . . . . .	QUSRSYS	Library . . . . .	
Severity . . . . .	00	Delivery . . . . .	*BREAK
Type reply (if required), press Enter.			
From . . . . .	LSVALGAARD	07/24/00	16: 55: 31
<b>Hello World</b>			



## ***MI Functional Reference Manual***

The MI Functional Reference Manual is available online ([http://www.as400.ibm.com/tstudio/tech\\_ref/mi/](http://www.as400.ibm.com/tstudio/tech_ref/mi/)). You can also order it from IBM. It is a very large document (1440 pages), but it is a must-have. The price is about \$100.00 depending on what version you want. There is very little difference between versions, so your best deal is to take the latest one you can get. Here is how to order it:

Go to <http://www1.ibm.link.ibm.com/>  
Select IBMLink for the United States.  
Select the PubsCenter link.  
Choose United States as Country  
Select Search for Publications  
On the Search screen, enter “Machine Interface”  
Click the Go button.  
You should get a list of the publications you are looking for.

This information is subject to change at IBM’s whim.

## Chapter 2

### Data Types and Your First *Real* MI-Program

#### Data Types

When you think about machine-level programming for a conventional computer, you may think in terms of assembly language with registers, addresses, and explicit type manipulation (e.g. different instructions for adding binary halfwords and floating point numbers). The AS/400 is radically different at the MI-level, although not at the RISC-level. At the MI-level, there are no accessible registers (some other - older - computer systems didn't have registers either), no addresses (here the AS/400 is unique, although some computers worked with 'descriptors'), and many instructions are polymorphic (know what to do based on the type of their operands). What are being manipulated in MI-programs are *primitive data-items* and *objects*. Primitive data-items can be referred to directly as operands in instructions. Objects are represented by *pointers*. Primitive data-items can also be referenced by pointers to them. We are not trying to be complete this early in the book and some of the finer details will be introduced further on.

#### Character Data Type

In MI, all operands must be *declared* with a declare statement **DCL**. Primitive data-items are declared using the Data Definition modifier DD: **DCL DD**. The most common data type is the character string. Here is how to declare the 30-character string **MY-TEXT**:

```
DCL DD MY-TEXT CHAR(30);
```

We can also give it an initial value:

```
DCL DD MY-TEXT CHAR(30) INIT("Hello");
```

A question arises here: the data-item is 30 characters long, will the **INIT** clause only initialize the first five characters (the length of "Hello") or will something else happen? This is very easy to test. Let's modify our **HELLO** program as follows:

```
DCL DD MY-TEST1 CHAR(30);
DCL DD MY-TEST2 CHAR(30) INIT("Hello");

CPYBLA      MSG-TEXT, MY-TEST1;
CALLI       SHOW-MESSAGE, *, . SHOW-MESSAGE;

CPYBLA      MSG-TEXT, MY-TEST2;
CALLI       SHOW-MESSAGE, *, . SHOW-MESSAGE;

RTX         *;
```

```
%I INCLUDE SHOWMSG
```

Note, that although MI is completely free-form, there is a benefit to sensible indentation and use of white space. Compile and run it:

Display Messages			
Queue . . . . .	LSVALGAARD	System:	AS400
Library . . . . .	QUSRSYS	Program . . . . .	*DSPMSG
Severity . . . . .	00	Library . . . . .	
		Deliver . . . . .	*BREAK
Type reply (if required), press Enter.			
From . . . . .	LSVALGAARD	07/25/00	15: 18: 22
From . . . . .	LSVALGAARD	07/25/00	15: 18: 25
Hello			

As you can see, the **INIT** clause initializes the remainder of the data-item to spaces (blanks). We can learn more from this simple test. **MSG-TEXT** is longer than 30 characters (in fact, it is 70 characters long), so the **CPYBLA** instruction (*without* the **P** for padding) copies as many characters as given by the length of the source (second) operand to the destination and leaves the rest of the destination unchanged. When you are

in doubt as to what a particular instruction or clause does, don't use it in a program you are writing hoping things will work out. Test it first by modifying the **HELLO** program just as we did above.

When you use quotes in literal strings, you can use either single quotes (') or double quotes ("), although they must match as a pair, so "Hello" is not valid, but "a single ' quote" and "a double "" quote" are fine, the latter having the same value as "a double "" quote".

## Numeric Data Types

MI supports the following numeric data types (each shown as an example):

```
DCL DD MY-HALFWORD BIN(2) INIT(10000); /* 2 byte halfword (16 bits) */
DCL DD MY-FULLWORD BIN(4) INIT(1000000); /* 4 byte fullword (32 bits) */
DCL DD MY-SHORT-FLOAT FLT(4) INIT(F' 3.1416'); /* 4 byte floating point */
DCL DD MY-LONG-FLOAT FLT(8) INIT(E' -1.5E+05'); /* 8 byte floating point */
DCL DD MY-ZONED ZND(8, 2) INIT(Z' -123.45'); /* 8 digit zoned with 2 decimals */
DCL DD MY-PACKED PKD(9, 2) INIT(P' +123.45'); /* 9 digit packed with 2 decimals */
```

As MI is "old" technology, there is no BIN(8) data type denoting a doubleword integer, although the 64-bit RISC processors naturally support these in hardware. Binary numbers can be unsigned. You declare that by using the UNSGND clause, as in:

```
DCL DD MY-UNSIGNED-HALFWORD BIN(2) UNSGND INIT(50000); /* 2 byte unsigned (16 bits) */
```

Note, that data-item names can be long. Note also, the peculiar way you have to specify initial values of floating-point, zoned, and packed data-items. This is related to the way you specify constants in instructions. The compiler needs to know what type the constant has. For packed and zoned data-items the first number, e.g. in PKD(15, 5) is the *total* number of digits including the decimals and the second number after the comma, the 5, is how many of those digits are to be used for the digits after the decimal point. So in the example, the largest value our PKD(15,5) data-item can represent is '9999999999.99999' This should be of no surprise to RPG and CL-programmers.

To try out some of this, modify our (ever-useful) **HELLO** program to read:

```
DCL DD MY-HALFWORD BIN(2) INIT(1000);
DCL DD MY-FULLWORD BIN(4) INIT(1000000);
DCL DD MY-ZONED ZND(10, 2); /* nnnnnnnn.dd */

ADDN      MY-ZONED, MY-HALFWORD, MY-FULLWORD;
CPYBLAP   MSG-TEXT, MY-ZONED, " ";
CALLI     SHOW-MESSAGE, *, . SHOW-MESSAGE;

RTX      *;

%INCLUDE SHOWMSG
```

I used the **ADDN** (Add Numeric) instruction to compute **MY-ZONED = MY-HALFWORD + MY-FULLWORD**. The fact that the three operands are of different types is no problem at all because the **ADDN** instruction is polymorphic (knows how to handle different types). To see the result, we now copy the result to **MSG-TEXT** and send the message as before. Because a zoned number has one digit per character, copying the result as a character string with **CPYBLAP** will work just fine. Here is what we get:

From . . . . : LSVALGAARD 07/25/00 20: 44: 00 0100100000
---

Which after positioning the decimal point in the correct place, is 100100.00, just as we would expect.

## Data Names

Although reserved words (like DCL, INIT, and instruction mnemonics) have to be UPPER case, you have more freedom with data names. A name can be up to 48 characters long and consist of any sequence of most of the printable characters, *except* the ones in the following set:

*blank / , ; ( ) : < + ' " %*

In addition, the *first* character of a name *cannot* be one of

- 0 1 2 3 4 5 6 7 8 9

Here are some valid names:

```
Thi s l sAVeryLongNameUsi ngMi xedCaseWords
A-COBOL-STYLE-NAME
#RGPNM
CUSTOMER_NAME_12
. PROGRAM-OBJECT
```

Names that begin with a period (.) are not inserted into the symbol table and cannot be referenced by the AS/400 debuggers. I personally use such names to signify *pointers* because of their readability (I never want to follow pointer chains with the debugger anyway). Names should neither be too long nor too short and cryptic. A good name is one you can read aloud over the telephone. Nothing new here.

## Comments

Comments can occur anywhere except in literal values and inside other comments, so cannot be nested. As we have seen, comments are bracketed by the special delimiters “/\*” and “\*/”, just like in CL-programs. A comment can span several lines. Beware of “run-away” comments where the ending delimiter “\*/” is missing. Comments are treated as blanks, so they are significant in delimiting names and tokens.

## Hexadecimal Constants

Character strings, binary integers, and floating-point values can be specified in hexadecimal notation, albeit in a different way for each:

```
DCL DD MY-TEXT      CHAR(7)  I N I T(X' C1C2C3C4C500' );      /* ABCDEnu l l */
DCL DD MY-BI NARY    BI N(2)   I N I T(H' 07D0' );            /* 2000 */
DCL DD MY-SHORT-FLOAT FLT(4)   I N I T(XF' BFC00000' );        /* -1.5 */
DCL DD MY-LONG-FLOAT FLT(8)    I N I T(XE' BFF8000000000000' ); /* -1.5 */
```

Binary numbers are right-justified with zero-fill on the left, so H' 01' is the same as H' 0001', which is, of course, just the BI N(2) representation of the number 1. As before, whenever in doubt about what something will do (or won't do) or you just want to explore something, modify the **HELLO** program:

```
DCL DD MY-HALFWORD1  BI N(2)   I N I T(H' 01' ); /* Is th i s H' 0100' or H' 0001' ? */
DCL DD MY-HALFWORD2  BI N(2)   I N I T(H' 0001' );
DCL DD MY-TEXT      CHAR(2);

CPYBLA      MY-TEXT, MY-HALFWORD1; /* make copy to character type */
CVTHC      MSG-TEXT(1: 4), MY-TEXT; /* convert to hexadecimal form */
CALLI      SHOW-MESSAGE, *, . SHOW-MESSAGE;

CPYBLA      MY-TEXT, MY-HALFWORD2;
CVTHC      MSG-TEXT(1: 4), MY-TEXT;
CALLI      SHOW-MESSAGE, *, . SHOW-MESSAGE;
. . .
```

The three periods simply mean that the remaining boilerplate is understood, viz.:

```
RTX      *;
%I NCLUDE SHOWMSG
```

## Convert Hexadecimal to Character

Notice the new instruction: **CVTHC** MSG-TEXT(1: 4), MY-TEXT. If we had just copied the binary value to **MSG-TEXT** with CPYBLA MSG-TEXT, MY-HALFWORD1 the message would contain non-printable characters and we wouldn't be able to tell the result. What we need is an instruction that converts the binary value into a string of hexadecimal digits that we can immediately understand. This is precisely what the **CVTHC**-instruction does. Its name (Convert Hexadecimal to Character) is somewhat misleading, as it really converts an arbitrary character string into its hexadecimal representation (i.e. goes the other way, “ABC” -> X'C1C2C3'). The instruction does require a character data-item and not a numeric data-item; that is why we had to first copy the numeric item to a character item. If you run the program, you'll see that the two values are the same. Yet another example of how you can easily try things out (promise: this is the last time I'll harp on that).

Another thing that the CVTHC-instruction insists on, is that the receiver must be exactly twice as long as the source, so we need a character item 4 bytes long to hold the hexadecimal representation of a 2-byte value. However, this isn't that strange if you think about it. If you look at the example where the three bytes for "ABC" expands to 6 bytes for X'C1C2C3' you can see the reason why.. You can accomplish that by a substring notation: MSG-TEXT(1: 4), meaning the substring starting in position 1 and extending 4 characters to the right.

## Compile Errors

Now is the time to deal with the problem of errors. You *will* make errors (I make many each and every day), so here goes. My first version of showing the hexadecimal value of a binary number looked like this:

```
DCL DD MY-HALFWORD BIN(2) INIT(H' 01');
DCL DD MY-TEXT CHAR(2);

      CVTHC      MSG-TEXT(1: 4), MY-HALFWORD;
CALLI      SHOW-MESSAGE, *, . SHOW-MESSAGE;
. . .
```

When compiling it, I got this message:

Processing command failure, refer to job log for details.

The job log contained:

Intermediate representation of program (IRP) contains 1 errors. Probable  
**compiler error.**

No, it's not a misprint or typo. The MI-translator (to use the proper term) was designed to process output from the HLL compilers, so if there were errors in the MI-statements, there probably *was* an error in the HLL compiler.

The MI-translator listing is spooled to **QSYSPT**:

5769SS1 V4R4M0 990521	Generated Output
SEQ INST Offset      Generated Code      *... .. 1 . . . . . 2 . . . . . 3 . . . . . 4 . . . . . 5 .	
00001	DCL DD MY-HALFWORD BIN(2) INIT(H' 0001')
00002	DCL DD MY-TEXT CHAR(2)
00003 <b>0001</b> 000004      1086 600B 2001 2004	CVTHC MSG-TEXT(1: 4), <b>MY-HALFWORD</b>
00004      0002 00000E      0293 001C 0000 001B	CALLI SHOW-MESSAGE, *, . SHOW-MESSAGE
00005      0003 000016      22A1 0000	RTX *
	/* INCLUDE: SHOWMSG */
. . . (lines omitted)	
MSGID      ODT      ODT Name      Semantics and ODT Syntax Diagnostics	
MSGID      MI Instruction Stream Semantic Diagnostics	
* CPF6412 <b>Attributes of Instruction X'0001' operand 2 not valid.</b>	

Clearly, a numeric data-item was not valid. It is not every time that the MI-compiler (back to a more traditional name) is that nice, sometimes there is no clue to *where* the error is, and sometimes there is even no clue to *what* the error is. It is therefore a good idea to code a little, try it, code a little more, try it, etc. Errors are then most likely in what you just added. Don't type in pages and pages before compiling.

## Conditions and Branching

Testing a condition and branching to a different point in the program depending on that condition is a fundamental operation. MI is very rich in this respect. Let us start with a simple loop executing 5 times; each time around the loop we simply display the loop counter:

```
DCL DD LOOP-COUNTER ZND(6, 0);      /* zoned number for ease of display      */

      CPYNV      LOOP-COUNTER, 0;      /* Copy Numeric Value 0 to LOOP-COUNTER */
LOOP:      /* a label, LOOP      */
      ADDN      LOOP-COUNTER, LOOP-COUNTER, 1;      /* LOOP-COUNTER = LOOP-COUNTER + 1      */
      CPYBLAP      MSG-TEXT, LOOP-COUNTER, " ";      /* show LOOP-COUNTER      */
      CALLI      SHOW-MESSAGE, *, . SHOW-MESSAGE;
      CMPNV(B)      LOOP-COUNTER, 5/LO(L0OP);      /* if LOOP-COUNTER < 5, go to LOOP      */
. . .
```

**CPYNV** (Copy Numeric Value) is another polymorphic instruction that copies the second numeric operand of any type to the first numeric operand of any type, with automatic type conversion (here from binary 0 to zoned 0000000). Then we add 1 to LOOP-COUNTER and show the result. Nothing new here. Except that one could gripe about the inconsistency of the instruction mnemonics, CPYNV has a “V” but ADDN has not. This hits me at least once a week.

The crucial instruction is the polymorphic Compare Numeric Value: **CMPNV(B)**, that compares the zoned LOOP-COUNTER with an immediate binary value 5 and sets *one* of four conditions:

- **LO** (if the first operand is lower than the second),
- **HI** (if it is higher),
- **EQ** (if they are equal),
- **UNOR** (“unordered”, if any of the operands is an invalid floating-point number, a so-called NaN – Not a Number).

A branch extender or modifier (B) to the operation code then directs the machine to execute one of more branches depending on the condition. In our case, we want to re-execute the loop if the condition is **LO** (we have not reached the end). The syntax for this is: **CMPNV(B) LOOP-COUNTER, 5/LO(LOOP)**. You can branch on several conditions in the same instruction:

**CMPNV(B) LOOP-COUNTER, 5/LO(LOOP), HI (ERROR), EQ(DONE)**

You reverse a condition by prefixing it with the letter **N** (for Not), e.g. **/NLO(GREATER-EQUAL)**.

## Labels and Branch Points

Labels are names followed by a colon (:). A point in the program marked with a label (labeled as is it) is called a *branch point*. You can *only* branch (goto, jump,...) to a labeled branch point. The standard error on other systems where you branch into the middle of some data and your program goes down (and maybe takes the system with it) cannot occur on an AS/400. Well, let’s say it is much *harder* to make that happen.

## Short Form of Instructions

Several instructions allow a shortened syntax if two operands are the same, e.g.:

**ADDN**            **LOOP-COUNTER, LOOP-COUNTER, 1; /\* LOOP-COUNTER = LOOP-COUNTER + 1 \*/**

can be shortened with the modifier **(S)** to:

**ADDN(S)        LOOP-COUNTER, 1;                        /\* LOOP-COUNTER = LOOP-COUNTER + 1 \*/**

You can combine the short modifier with the branch modifier as in the following example that uses the Subtract Numeric (SUBN) instruction to execute the loop counting the LOOP-COUNTER *down* from 5 to 0:

```
CPYNV            LOOP-COUNTER, 5;
LOOP:
CPYBLAP        MSG-TEXT, LOOP-COUNTER, " ";
CALLI           SHOW-MESSAGE, *, . SHOW-MESSAGE;
SUBN(SB)        LOOP-COUNTER, 1/POS(LOOP);        /* if positive ( > 0 ) loop again */
```

Here, the decrement, the test, and the branch are all handled by the same instruction. The preferred condition names for this instruction are:

- **POS** (result>0),
- **NEG** (result<0),
- **ZER** (result=0),

but **HI**, **LO**, and **EQ** work just the same. The order of the instruction-modifiers does not matter, so for example SUBN(SB) is the same as SUBN(BS), although it is best to stick to the same order throughout.

## Structured Data

In RPG and COBOL we are familiar with the concept of a *record*, that contains various *fields*. We say that the record contains structured data. Consider the following COBOL record:

```

01 MY-RECORD.
02 MY-BINARY          PIC S9(9)    BINARY.
02 FILLER              PIC X(1)     VALUE "=".
02 MY-PACKED           PIC S9V9(8)  PACKED.
02 FILLER              PIC X(6)     VALUE SPACES.

```

It has a direct counterpart in MI as follows:

```

DCL DD MY-RECORD CHAR(16);
DCL DD MY-BINARY      BIN(4) DEF(MY-RECORD) POS( 1);
DCL DD *              CHAR(1) DEF(MY-RECORD) POS( 5) INIT("=");
DCL DD MY-PACKED      PKD(9,8) DEF(MY-RECORD) POS( 6);
DCL DD *              CHAR(6) DEF(MY-RECORD) POS(11) INIT(" ");

```

Fields are defined (**DEF**) on the record starting in the character position stated (**POS**). The special name “\*” means “no name” or “FILLER” in COBOL. We can now assign values to the fields and move the record as a unit:

```

CPYNV      MY-PACKED, P' 3.14159265';
CPYNV      MY-BINARY, MY-PACKED;
CVTHC      MSG-TEXT(1:32), MY-RECORD;
CALLI      SHOW-MESSAGE, *, . SHOW-MESSAGE;

```

with this result: 000000037E314159265F404040404040

Note, that the moving 3.14159265 to the binary integer yields (as we expect) the number 3.

Here is a somewhat more complicated structure (that we’ll use in a later section):

```

DCL DD MACHINE-ATTR CHAR(256) BDRY(16);
DCL DD BYTES-PROVIDED BIN(4) DEF(MACHINE-ATTR) POS(1) INIT(256);
DCL DD BYTES-AVAILABLE BIN(4) DEF(MACHINE-ATTR) POS(5);
DCL DD THE-ATTRIBUTES CHAR(248) DEF(MACHINE-ATTR) POS(9);

DCL DD THE-TIMESTAMP CHAR(8) DEF(THE-ATTRIBUTES) POS( 1);
DCL DD THE-TIME-HI BIN(4) UNSGND DEF(THE-TIMESTAMP) POS(1);
DCL DD THE-TIME-LO BIN(4) UNSGND DEF(THE-TIMESTAMP) POS(5);

DCL DD SERIAL-NBR CHAR(8) DEF(THE-ATTRIBUTES) POS( 1);

DCL DD NETWORK-ATTRS CHAR(190) DEF(THE-ATTRIBUTES) POS( 1);
DCL DD SYSTEM-NAME CHAR(8) DEF(NETWORK-ATTRS) POS( 1);
DCL DD * BIN(2) DEF(NETWORK-ATTRS) POS( 9);
DCL DD NEW-SYSTEM-NAME CHAR(8) DEF(NETWORK-ATTRS) POS(11);
DCL DD * BIN(2) DEF(NETWORK-ATTRS) POS(19);
DCL DD LOCAL-NETWORK-ID CHAR(8) DEF(NETWORK-ATTRS) POS(21);
DCL DD * BIN(2) DEF(NETWORK-ATTRS) POS(29);

```

Note, how THE-TIME-HI is defined on THE-TIMESTAMP, which in turn is defined on THE-ATTRIBUTES, which in turn is defined on MACHINE-ATTR. Note also, how the substructures THE-TIMESTAMP, SERIAL-NBR and NETWORK-ATTRS are all redefinitions of THE-ATTRIBUTES.

Again, the best way to describe the above data structure is in COBOL:

```

01 MACHINE-ATTR.
02 BYTES-PROVIDED          PIC S9(9)    BINARY VALUE 256.
02 BYTES-AVAILABLE        PIC S9(9)    BINARY.
02 THE-ATTRIBUTES         PIC X(248).

02 THE-TIMESTAMP          REDEFINES THE-ATTRIBUTES.
03 THE-TIME-HI            PIC 9(9)     BINARY.
03 THE-TIME-LO            PIC 9(9)     BINARY.

02 SERIAL-NBR             REDEFINES THE-ATTRIBUTES.
                           PIC X(8).

02 NETWORK-ATTRS          REDEFINES THE-ATTRIBUTES.
03 SYSTEM-NAME            PIC X(8).
03 FILLER                 PIC S9(4)     BINARY.
03 NEW-SYSTEM-NAME        PIC X(8).
03 FILLER                 PIC S9(4)     BINARY.
03 LOCAL-NETWORK-ID      PIC X(8).
03 FILLER                 PIC S9(4)     BINARY.

```

## Our First “Real” MI-Program

Using the structure we have just defined, we shall now proceed to write our first “real” MI-program, defined as one that does something useful, instead of just serving as a learning tool as our venerable **HELLO** program did. There is a very useful MI-instruction called Materialize Machine Attributes (**MATMATR**). As you can guess, this instruction gives us all kinds of information about the machine you are running on. The description of this single instruction in the MI Functional Reference Manual (see later, how to get one) takes up 34 pages of dense text! The instruction is coded with two operands, basically like this:

```
MATMATR      .MACHINE-ATTR, MATMATR-CONTROL;
```

The second operand, **MATMATR-CONTROL**, is simply a 2-character control value whose contents determine what the instruction “materializes”. You will see that very many MI-instructions use the concept of *materializing* information from internal sources and making it available to you in a suitable data structure, that is often called a *template*. The first operand is not the template, but a *pointer* to the template. We’ll explore the concept of a pointer in great detail later. For now, all we need is how to make a pointer point to our data structure. The kind of pointer that simply points to some data is called a *space pointer*. You declare a space pointer by using the **SPCPTR** modifier and you can in the declare-statement directly initialize the pointer to point to the desired data structure, like this:

```
DCL SPCPTR .MACHINE-ATTR INIT(MACHINE-ATTR);  
DCL DD MACHINE-ATTR CHAR(256) BDRY(16);
```

For reasons we’ll come to later, this template has to *aligned* on a 16-byte boundary, indicated by the **BDRY(16)** clause. Note our naming convention of having a pointer to a data-item have the same name as the item, except prefixed by a period. Other people have conventions that use a prefix of “?”, “@”, or as in C: “\*”.

We got the layout of the template from the MI Functional Reference (maybe we should invent a new acronym: MIFR). The MIFR also lists the value of the control parameter. The values we are interested in are:

X' 0004'	Get the machine Serial Number
X' 0100'	Get the Internal Clock value
X' 0130'	Get Network Attributes

Declare the control operand as:

```
DCL DD MATMATR-CONTROL CHAR(2);
```

The program code is now straightforward:

```
CPYBLA      MATMATR-CONTROL, X' 0004'; /* Get Serial Number */  
MATMATR     .MACHINE-ATTR, MATMATR-CONTROL;  
CPYBLAP     MSG-TEXT, SERIAL-NBR, " "; /* Text */  
CALLI      SHOW-MESSAGE, *, .SHOW-MESSAGE;  
  
CPYBLA      MATMATR-CONTROL, X' 0100'; /* Get Timestamp */  
MATMATR     .MACHINE-ATTR, MATMATR-CONTROL;  
CVTHC      MSG-TEXT(1:16), THE-TIMESTAMP; /* Binary */  
CALLI      SHOW-MESSAGE, *, .SHOW-MESSAGE;  
  
CPYBLA      MATMATR-CONTROL, X' 0130'; /* Get Network Attrs */  
MATMATR     .MACHINE-ATTR, MATMATR-CONTROL;  
CPYBLAP     MSG-TEXT(1:30), NETWORK-ATTRS, " "; /* mixed */  
CALLI      SHOW-MESSAGE, *, .SHOW-MESSAGE;  
.  
.  
.
```

The result should be similar to this:

From . . . :	LSVALGAARD	07/26/00	21:46:57
1234567		<=	serial number
810566576CDA8000		<=	current internal time value
AS400	APPN	<=	system name, etc

For completeness, we show the entire program (**MI MCHATTR**) on a single page below. Copy and paste it to a file, transfer it to the AS/400, compile and run it.



```

/* Materialize Machine Attributes */
DCL DD MATMATR-CONTROL CHAR(2);
DCL SPCPTR .MACHINE-ATTR INIT(MACHINE-ATTR);

DCL DD MACHINE-ATTR CHAR(256) BDRY(16);
DCL DD BYTES-PROVIDED BIN(4) DEF(MACHINE-ATTR) POS(1) INIT(256);
DCL DD BYTES-AVAILABLE BIN(4) DEF(MACHINE-ATTR) POS(5);
DCL DD THE-ATTRIBUTES CHAR(248) DEF(MACHINE-ATTR) POS(9);

DCL DD THE-TIMESTAMP CHAR(8) DEF(THE-ATTRIBUTES) POS( 1);
DCL DD THE-TIME-HI BIN(4) UNSGND DEF(THE-TIMESTAMP) POS(1);
DCL DD THE-TIME-LO BIN(4) UNSGND DEF(THE-TIMESTAMP) POS(5);

DCL DD SERIAL-NBR CHAR(8) DEF(THE-ATTRIBUTES) POS( 1);

DCL DD NETWORK-ATTRS CHAR(190) DEF(THE-ATTRIBUTES) POS( 1);
DCL DD * SYSTEM-NAME CHAR(8) DEF(NETWORK-ATTRS) POS( 1);
DCL DD * BIN(2) DEF(NETWORK-ATTRS) POS( 9);
DCL DD NEW-SYSTEM-NAME CHAR(8) DEF(NETWORK-ATTRS) POS(11);
DCL DD * BIN(2) DEF(NETWORK-ATTRS) POS(19);
DCL DD LOCAL-NETWORK-ID CHAR(8) DEF(NETWORK-ATTRS) POS(21);
DCL DD * BIN(2) DEF(NETWORK-ATTRS) POS(29);

CPYBLA MATMATR-CONTROL, X'0004'; /* Get Serial Number */
MATMATR .MACHINE-ATTR, MATMATR-CONTROL;
CPYBLAP MSG-TEXT, SERIAL-NBR, " "; /* Text */
CALLI SHOW-MESSAGE, *, .SHOW-MESSAGE;

CPYBLA MATMATR-CONTROL, X'0100'; /* Get Timestamp */
MATMATR .MACHINE-ATTR, MATMATR-CONTROL;
CVTHC MSG-TEXT(1:16), THE-TIMESTAMP; /* Binary */
CALLI SHOW-MESSAGE, *, .SHOW-MESSAGE;

CPYBLA MATMATR-CONTROL, X'0130'; /* Get Network Attrs */
MATMATR .MACHINE-ATTR, MATMATR-CONTROL;
CPYBLAP MSG-TEXT(1:30), NETWORK-ATTRS, " "; /* mixed */
CALLI SHOW-MESSAGE, *, .SHOW-MESSAGE;

RTX *;

%INCLUDE SHOWMSG

```

## Chapter 3

### Pointers, Pointers, and Pointers

#### Space Data Object

In chapter 2 we developed a program (**MI MCHATR**) to materialize machine attributes into a structured character data-item called a template. The various data-items were declared to be in specific positions relative to the beginning of the area. Creating and (especially maintaining) such a sequence of absolute positions is tedious and prone to errors. There is a much more elegant way of specifying a data structure by using a *space data object*, which is defined as a *based* (i.e. beginning at a location in memory identified by a pointer) 32,767-character string. What in effect we are doing is telling the MI-compiler that no matter what the data ‘looks’ like in memory, at the location the pointer this structure is based upon points to, we want to map this layout over it so that we can access it using the structures data-items. A space data object is an internal program declaration of a storage mapping. No real space in memory is allocated for the structure, but the structure is laid over an actual character string determined by the pointer, allowing us to reference sections of the data by the data-item names we used in defining the data space object. Here is the declaration of the materialization area using a space data object (**DCL SPC**):

```
DCL DD MATERIALIZE-AREA CHAR(256) BDRY(16); /* This is the actual data-item */
DCL SPCPTR .MACHINE-ATTR INIT(MATERIALIZE-AREA); /* Points to actual data-item */
DCL SPC MACHINE-ATTR BAS(.MACHINE-ATTR); /* based upon the pointer */
DCL DD BYTES-PROVIDED BIN(4) DIR;
DCL DD BYTES-AVAILABLE BIN(4) DIR;

DCL DD THE-TIMESTAMP CHAR(8) DEF(MACHINE-ATTR) POS(9);
DCL DD THE-TIME-HI BIN(4) UNSGND DIR;
DCL DD THE-TIME-LO BIN(4) UNSGND DIR;

DCL DD SERIAL-NBR CHAR(8) DEF(MACHINE-ATTR) POS(9);

DCL DD NETWORK-ATTRS CHAR(190) DEF(MACHINE-ATTR) POS(9);
DCL DD SYSTEM-NAME CHAR(8) DIR;
DCL DD * BIN(2) DIR;
DCL DD NEW-SYSTEM-NAME CHAR(8) DIR;
DCL DD * BIN(2) DIR;
DCL DD LOCAL-NETWORK-ID CHAR(8) DIR;
DCL DD * BIN(2) DIR;
```

If you compare this with the previous definition, you will appreciate the simplicity gained. There is no need to tediously count character positions as the “direct” (**DIR**) clause instructs the compiler to automatically increment its internal position counter from its previous value, starting at 1 and increasing this internal position counter with the size of each item. You can still, if needed, specify a **POS** clause to explicitly change the position counter. We did this thrice to specify that **THE-TIMESTAMP**, **SERIAL-NBR**, and **NETWORK-ATTR** all start at the same position (and thus overlay each other). Because a space data object only maps into the real area and could map into different areas by changing the pointer, we cannot initialize data-items in the map, so the following would be an incorrect definition;

```
DCL SPCPTR .MACHINE-ATTR INIT(MATERIALIZE-AREA); /* Points to actual data-item */
DCL SPC MACHINE-ATTR BAS(.MACHINE-ATTR); /* based upon the pointer */
DCL DD BYTES-PROVIDED BIN(4) DIR;
DCL DD BYTES-AVAILABLE BIN(4) DIR;

DCL DD THE-TIMESTAMP CHAR(8) DIR POS(9);
DCL DD THE-TIME-HI BIN(4) UNSGND DIR INIT( X'0101' ); /*<- Incorrect */
DCL DD THE-TIME-LO BIN(4) UNSGND DIR;
...
```

But we can do it in the actual data-item being pointed to, in the instruction section of the program:

```
CPYNV BYTES-PROVIDED, 256;
```

The rest of the code stays the same.

## Setting a Space Pointer

While the `INIT` clause can establish addressability to a data-item in a space pointer declaration, like the one we used above: `DCL SPCPTR .MACHINE-ATTR INIT(MATERIALIZE-AREA)`, the connection, in this example that the space pointer `MACHINE-ATTR` is to point to the area of memory that the compiler has allocated for the `MATERIALIZE-AREA` data-item, is established once when the program starts. If we need to point to a different area, we need a way to set addressability in a space pointer dynamically under program control. The Set Space Pointer (**SETSPP**) instruction does just that. So, instead of initializing the pointer in the declaration, we could have written:

```
DCL DD MATERIALIZE-AREA CHAR(256) BDRY(16); /* This is the actual data-item. */
DCL SPCPTR .MACHINE-ATTR; /* Does not point to anything yet */

SETSPP .MACHINE-ATTR, MATERIALIZE-AREA; /* Only now can be use the data */
CPYNV BYTES-PROVIDED, 256; /* based on the pointer */
```

## Explore Pointers: Materialize Invocation Stack

This problem comes up regularly: how do I find out which program called the current program? As programs call each other an *invocation stack* is built up. You can see the contents of the stack using the **DSPJOB** command, selection 11:

Rqs Lvl	Program or Procedure	Library	Statement	Instruction
	QCMD	QSYS		043B
	QUI CMENU	QSYS		00C1
1	QUI MNDRV	QSYS		0502
2	QUI MGFLW	QSYS		04B5
3	QUI CMD	QSYS		0419
	MI INVSTK	LSVALGAARD		000F

This problem is well suited to explore various pointer types and their use and is non-trivial. A quick look though the MIFR table of contents finds an MI instruction named **MATINVS** (MATERialize INVocation Stack) that looks to be a prime candidate to help us achieve a solution. The **MATINVS** MI-instruction takes as its first operand a space pointer to the materialization area and as its second operand a new kind of pointer that we haven't covered yet, called a *system pointer*, to the job for which to retrieve the invocation stack. At this point we are only interested in our own job. Using a *null* operand, `""`, selects our own job. So this is the instruction we shall use:

```
MATINVS .THE-STACK, *; /* OWN JOB */
```

The stack has no set depth, so we don't know how many programs will be on it. There are two methods we could use static memory or dynamically allocated memory.

## Static Storage Method

One way (which we shall use initially) is to allocate a fixed area of a reasonably large size, say fifty programs. As before, the template begins with two binary numbers, the first giving the size of the materialization area, the second will be set to how many bytes were actually returned ("materialized") by the instruction. The number of entries on the stack is returned as a binary number in the third position. The list of entries is returned in the materialization area. Because each entry of the list contains pointers and because pointers (for historical reasons) must be aligned on 16-byte boundaries, the entire area must also be aligned on a 16-byte boundary. Finally, the list within the area must also be aligned on a 16-byte boundary. The net effect of all these alignment requirements is a 4-byte "hole" or slack-area. In descriptions of the MI-instructions in the MIFR, such holes are often designated as "reserved". When creating data definitions in MI programs, such unused space is conventionally named `""`. You could also simply omit any reference to it, if data-item following the unused space is defined by its explicit position using the `POS(n)` clause. Here is then the full declaration:

```
DCL SPCPTR .THE-STACK;
DCL DD THE-STACK CHAR(6416) BDRY(16);
DCL DD STK-BYTES-PRV BIN(4) DEF(THE-STACK) POS( 1);
DCL DD STK-BYTES-AVL BIN(4) DEF(THE-STACK) POS( 5);
DCL DD STK-NBR-OF-ENTRIES BIN(4) DEF(THE-STACK) POS( 9);
DCL DD * BIN(4) DEF(THE-STACK) POS(13);
```

```
DCL DD STK-ENTRY(50) CHAR(128) DEF(TH-STACK) POS(17);
```

Note how we declare an *array* of 50 entries. According to the MIFR, each entry must be 128 characters long. The exact layout of an invocation stack entry is given in all its gory details in the MIFR. What is of immediate interest to us is only the fact that a *system pointer* to the program occupying this position of the stack is stored starting in position 33 within each entry:

```
DCL DD TH-ENTRY CHAR(128) BDRY(16);
DCL SYSPTR .TH-ENTRY-PGM DEF(TH-ENTRY) POS(33);
```

To process the  $n$ 'th entry, we must either copy it to an entry holding area (like the one we just declared): `CPYBLA TH-ENTRY, STK-ENTRY( $n$ )`, or define a space data object based on a space pointer set dynamically to the  $n$ 'th entry: `SETSP .TH-ENTRY, STK-ENTRY( $n$ )` using this declaration instead:

```
DCL SPCPTR .TH-ENTRY;
DCL DD TH-ENTRY CHAR(128) BAS(.TH-ENTRY);
DCL SYSPTR .TH-ENTRY-PGM DEF(TH-ENTRY) POS(33);
```

To materialize the stack, we code:

```
GET-STACK:
  CPYNV      STK-BYTES-PRV, 6416;    /* area size provided */
  SETSP      .TH-STACK, TH-STACK;    /* point to the area */
  MATI NVS   .TH-STACK, *;           /* your own job */
```

We process the stack entries with a simple loop:

```
DCL DD PGM-NBR BIN(4);

  CPYNV      PGM-NBR, 0;
NEXT-PROGRAM:
  ADDN(S)    PGM-NBR, 1;
  CMPNV(B)   PGM-NBR, STK-NBR-OF-ENTRIES/HI (DONE);
  CPYBWP     .TH-ENTRY, STK-ENTRY(PGM-NBR);
  . . . /* process this entry */
  B          NEXT-PROGRAM;
DONE:
```

Since each entry returned contains at least one pointer, we need a special copy-instruction **CPYBWP** (Copy Bytes With Pointers). Pointers carry special *tag bits* that must be set for the pointer to be valid. The usual copy instructions do not preserve the tag bits, hence the special instruction. We shall deal extensively with the issue of tag bits later on.

Alternatively, we can use a space pointer to point dynamically to the current entry, and then access the entry through a space data object based on the pointer:

```
  CPYNV      PGM-NBR, 0;
NEXT-PROGRAM:
  ADDN(S)    PGM-NBR, 1;
  CMPNV(B)   PGM-NBR, STK-NBR-OF-ENTRIES/HI (DONE);
  SETSP      .TH-ENTRY, STK-ENTRY(PGM-NBR);
  . . . /* process this entry */
  B          NEXT-PROGRAM;
DONE:
```

In both cases, the code for processing the entry is the same. We want to show the name of the program object and the name of the library (called a *context* at the MI-level) where the program object is stored. The materialized entry returned from `MATI NVS` does not contain this information directly. Instead, we just get a system pointer to the program object, remember how we defined the data-item `.TH-ENTRY-PGM` as a system pointer (`SYSPTR`) at position 33 of `TH-ENTRY`?. A system pointer uniquely identifies an object in the system, but the identifier is in an internal binary format. The Materialize Pointer (**MATPTR**) instruction can return the *symbolic* identification (i.e. the type, name, and context) of the object identified by the pointer. As with all materialize-instructions we provide a space pointer to the area where we want the information to be returned:

```
DCL SPCPTR .PTR-TEMPLATE INIT(PTR-TEMPLATE);
DCL DD PTR-TEMPLATE CHAR(76);
DCL DD PTR-PROVIDED BIN(4) DEF(PTR-TEMPLATE) POS( 1);
DCL DD PTR-RETURNED BIN(4) DEF(PTR-TEMPLATE) POS( 5);
DCL DD PTR-TYPE CHAR(1) DEF(PTR-TEMPLATE) POS( 9);
```

```

DCL DD PTR-DATA          CHAR(68) DEF(PTR-TEMPLATE) POS(10);
DCL DD PTR-CTX-TYPE      CHAR( 2) DEF(PTR-DATA)      POS( 1);
DCL DD PTR-CTX-NAME      CHAR(30) DEF(PTR-DATA)      POS( 3);
DCL DD PTR-OBJ-TYPE      CHAR( 2) DEF(PTR-DATA)      POS(33);
DCL DD PTR-OBJ-NAME      CHAR(30) DEF(PTR-DATA)      POS(35);
DCL DD PTR-...           CHAR( 4) DEF(PTR-DATA)      POS(65);

```

First, we specify how much of the data that could be materialized we are really interested in, then we materialize the object information for THE-ENTRY-PGM identified by the system pointer returned in the stack entry:

```

CPYVNV      PTR-PROVIDED, 76;
MATPTR      . PTR-TEMPLATE, . THE-ENTRY-PGM;

```

## Dealing with Exceptions

If we run the program that we have put together so far (try it) we get a “hardware protection” error (MCH6801) at security levels above 30 (and you don’t run below, do you?). This is because some of the programs in the stack are located in the *system domain*, and therefore are not accessible from a user-state program:

	QCMD	QSYS	user domain
	QUI CMENU	QSYS	system domain
1	QUI MNDRV	QSYS	system domain
2	QUI MGFLW	QSYS	system domain
3	QUI CMD	QSYS	system domain
	MI INVSTO	LSVALGAARD	user domain

We would like to monitor for this exception and deal gracefully with the situation, e.g. simply skip programs we are not allowed to see. You can declare an exception monitor (**EXCM**) to catch exceptions and direct the flow of the program to your exception handler. Here is how:

```

DCL EXCM * EXCID(H' 4401') BP(NEXT-PROGRAM) CV("MCH") IMD;

```

This monitor is unnamed (the “\*”) and takes effect immediately (the **IMD**) upon the exception being signaled transferring control to the branch point (the BP) NEXT-PROGRAM. The exception identifier has two parts, a hexadecimal value (the EXCID) and a so-called compare value (the CV). A tricky detail is that for MCH (Machine Check) exceptions, the value reported (e.g. MCH6801) is *not* what you should use as the exception identifier in the exception monitor. Instead, the first two digits and the last two digits must be converted from decimal to hexadecimal *separately*. Since the decimal value 68 = 6\*10+8 has the hexadecimal value 44 = 4\*16+4, we must monitor for MCH4401 to detect MCH6801. I have given up on trying to see the logic behind this and have accepted that that’s the way it works. For CPF and other exceptions, no such conversion is needed as they are already in hex format; only MCH exceptions have this quirk. This difference is constant pain.

We can now complete the processing to show the value of the pointer and the names of the context and program using our standard show-message routine:

```

CPYVNV      PGM-NBR, 0;
NEXT-PROGRAM:
ADDN(S)     PGM-NBR, 1;
CMPNV(B)    PGM-NBR, STK-NBR-OF-ENTRIES/HI (DONE);
CPYBWP      THE-ENTRY, STK-ENTRY(PGM-NBR);

CPYVNV      PTR-PROVIDED, 76;
MATPTR      . PTR-TEMPLATE, . THE-ENTRY-PGM;

CPYBREP     MSG-TEXT, " ";
CVTHC      MSG-TEXT(1:32), THE-ENTRY(33:16);

CPYBLAP     MSG-TEXT(34:11), PTR-CTX-NAME, " ";
CPYBLAP     MSG-TEXT(46:11), PTR-OBJ-NAME, " ";
CALLI      SHOW-MESSAGE, *, . SHOW-MESSAGE;
B          NEXT-PROGRAM;

DONE:
RTX        *;

%I INCLUDE SHOWMSG

```

Here is a typical result (note that only the user domain programs are showing):

```
From . . . . : LSVLGAARD      08/03/00   20: 20: 43
000000000000000002433117B98000200 QSYS      QCMD
00000000000000000067C6EE75F000200 LSVLGAARD  MI INVSTK
```

## Automatic Storage Allocation

We had allocated an array of 50 entries for the materialized invocation stack. The materialized data was part of the *static storage* that is allocated when the program is first called. If we do not wish to be limited to a predefined number of entries, it is possible to allocate the stack in *automatic storage* and to modify its size dynamically as needed. The first step is to declare the stack with the **AUTO** clause with the smallest possible size (8 characters) holding only the number of bytes provided and the number of bytes available for materialization:

```
DCL SPCPTR .THE-STACK;
DCL DD     THE-STACK CHAR(8) AUTO BDRY(16);
DCL DD     STK-BYTES-PRV BIN(4) DEF(THE-STACK) POS( 1);
DCL DD     STK-BYTES-AVL BIN(4) DEF(THE-STACK) POS( 5);
```

We then set the number of bytes provided to cover the minimum size, set a space pointer to point to the materialization area, and materialize the stack:

```
CPYNV      STK-BYTES-PRV, 8;          /* Minimum */
SETSP      .THE-STACK, THE-STACK;    /* Point to the receiver area */
MATINVS    .THE-STACK, *;            /* Materialize the minimum */
```

As result we get the number of bytes needed, STK-BYTES-AVL, to receive all entries in the stack. The Modify Automatic Storage instruction, **MODASA**, can then be used to change the size of the area pointed to:

```
MODASA     .THE-STACK, STK-BYTES-AVL; /* Change size of area */
```

Because this operation may allocate a different area of storage, we need to repeat the instruction that sets a space pointer to point to the area, and then finally materialize the invocation stack again::

```
SETSP      .THE-STACK, THE-STACK;    /* Point to changed area */
MATINVS    .THE-STACK, *;            /* Materialize full stack */
```

To access the entries, we can define an array as before. Since we cannot define a variable size array, we define the array with size 1, and then switch off subscript checking (using the Override Program Attribute instruction, OVRPGATR):

```
DCL DD     STK-NBR-OF-ENTRIES BIN(4) DEF(THE-STACK) POS( 9);
DCL DD     STK-ENTRY(1) CHAR(128) DEF(THE-STACK) POS(17);

OVRPGATR   1, 2; /* Do not constrain array references */
```

To summarize:

```
DCL SPCPTR .THE-STACK;
DCL DD     THE-STACK CHAR(8) AUTO BDRY(16);
DCL DD     STK-BYTES-PRV BIN(4) DEF(THE-STACK) POS( 1);
DCL DD     STK-BYTES-AVL BIN(4) DEF(THE-STACK) POS( 5);
DCL DD     STK-NBR-OF-ENTRIES BIN(4) DEF(THE-STACK) POS( 9);
DCL DD     STK-ENTRY(1) CHAR(128) DEF(THE-STACK) POS(17);

GET-STACK:
CPYNV      STK-BYTES-PRV, 8;          /* Minimum */
SETSP      .THE-STACK, THE-STACK;    /* Point to the receiver area */
MATINVS    .THE-STACK, *;            /* Materialize the minimum */

MODASA     .THE-STACK, STK-BYTES-AVL; /* Change size of area */

CPYNV      STK-BYTES-PRV, STK-BYTES-AVL; /* New size */
SETSP      .THE-STACK, THE-STACK;
MATINVS    .THE-STACK, *;

OVRPGATR   1, 2; /* Do not constrain array references */

CPYNV      PGM-NBR, 0;
NEXT-PROGRAM:
...
```

## Instruction Pointers

We have several times used the following instruction to call the SHOW-MESSAGE subroutine without really understanding its operands:

CALLI SHOW-MESSAGE, \*, . SHOW-MESSAGE;

The third operand **.SHOW-MESSAGE** is an *instruction pointer*. The **CALLI** instruction stores the return point in this pointer variable then goes to the **SHOW-MESSAGE** entry point which is **internal** to our program:

```

DCL  INSPTR  .SHOW-MESSAGE;      /* declare instruction pointer */
ENTRY .      .SHOW-MESSAGE INT;   /* other processing . . . */
      B      .SHOW-MESSAGE;      /* return to where it came from */

```

This may look very strange in the source code, and if you're not thinking in MI-mode when reading the source, it may look like the final branch statement of `SHOW-MESSAGE` will actually cause the `SHOW-MESSAGE` subroutine to be executed again causing an endless loop. However what really happens is that when `SHOW-MESSAGE` is finished, the subroutine returns using a branch-instruction (B) passing program control back to the return point instruction that is *pointed* to by . `SHOW-MESSAGE` pointer.

## Chapter 4

### Arithmetic and Timestamps

#### Timestamps

The AS/400 maintains an internal clock whose value is often used as a fine-grained timestamp. The clock value is 64 bits long and increments by 4096 every microsecond (or at least looks like it does as the exact way it counts may depend on the hardware implementation). The starting time (12:03:06.3148pm on August 23<sup>rd</sup>, 1928) of the counter was chosen such that the most significant bit would change (“roll over”) at the end of day on December 31<sup>st</sup>, 1999. In this chapter we shall develop a program to convert the timestamp into a more conventional text format: YYYYMMDDhhmmss. This allows us to explore several arithmetical instructions on various data types. We learned in Chapter 2 how to materialize the machine clock. The materialization area had this structure:

```
DCL DD MACHINE-CLOCK CHAR(2) INIT(X'0100');
DCL SPCPTR .MACHINE-ATTR INIT(MACHINE-ATTR);
DCL DD .MACHINE-ATTR CHAR(24) BDRY(16);
DCL DD MAT-MAX-SIZE BIN(4) DEF(MACHINE-ATTR) POS( 1) INIT(16);
DCL DD MAT-ACT-SIZE BIN(4) DEF(MACHINE-ATTR) POS( 5);
DCL DD MAT-TIMESTAMP CHAR(8) DEF(MACHINE-ATTR) POS( 9);
DCL DD MAT-TIME-HI BIN(4) UNSGND DEF(MAT-TIMESTAMP) POS(1);
DCL DD MAT-TIME-LO BIN(4) UNSGND DEF(MAT-TIMESTAMP) POS(5);
```

#### Convert Timestamp to Date and Time

The 8-character timestamp consists of two *unsigned* binary halves, a high-order part and a low-order part. We need to do this because MI does not support 8-byte binary values BIN(8). Instead, we shall use packed decimal arithmetic, so the first task is to convert the 8-byte binary value into a packed decimal number. Let us define the following variables:

```
DCL DD TIMESTAMP PKD(21,0); /* Can hold 64 bits unsigned */
DCL DD TIMESTAMP-HI PKD(11,0); /* Can hold 32 bits unsigned */
DCL DD TIMESTAMP-LO PKD(11,0); /* Can hold 32 bits unsigned */
DCL DD TWO**32 PKD(11,0) INIT(P'4294967296'); /* 232 */
```

Since we *can* copy the unsigned 32-bit binary halves of the timestamp to their corresponding packed values, `TIMESTAMP-HI` and `TIMESTAMP-LO`, we can easily compute the packed decimal full value of the timestamp:  $TIMESTAMP = TIMESTAMP-HI * 2^{32} + TIMESTAMP-LO$ :

```
CPYVNV .TIMESTAMP-LO, MAT-TIME-LO; /* copy unsigned binaries */
CPYVNV .TIMESTAMP-HI, MAT-TIME-HI;

MULT .TIMESTAMP, .TIMESTAMP-HI, TWO**32;
ADDN(S) .TIMESTAMP, .TIMESTAMP-LO;
```

Since the clock increments by 4096 every microsecond there are 4,096,000,000 “clicks” in one second. To compute the number of seconds that have elapsed since the starting time, we must divide by 4,096,000,000:

```
DIV(SR) .TIMESTAMP, 4096000000; /* Now seconds, rounded */
```

Because we had decided to work to an accuracy of one second only, we *rounded* the result using the **R** operation extender on the Divide instruction (**DIV**). Although the AS/400 internal clock starting time of 12:03:06.3148pm on August 23<sup>rd</sup>, 1928 gave a easy number to work with for the start of the year 2000, it also gives us a ugly number to use for date arithmetic, so we will make some initial adjustments to the timestamp returned by the `MATMATR` instruction internally in our program to make the mathematics a little easier. We can compensate for the awkward starting time within the day by adding the number of seconds from the previous midnight up to 12:03:06pm:

```
ADDN(S) .TIMESTAMP, 43386; /* 12:03:06pm */
```

We now have the number of seconds elapsed since the beginning of the starting day. To get the number of days, `NBR-DAYS`, that have elapsed, we divide by the number of seconds in a day, 86400, and retain the



remainder as the number of seconds, NBR-SECONDS, elapsed within the last day. The Divide with Remainder instruction (**DI VREM** *quotient, dividend, divisor, remainder*) does this handily:

```
DI VREM      NBR-DAYS, T I MESTAMP, 86400, NBR-SECONDS;
```

## Converting to a Date

Taking into account the intricacies of the calendar is eased by the fact that year 2000 is a leap year, so we don't need to use the 100-year rule. Again, to make the calculations easier, we can compensate for the awkward starting day by reducing the number of days by 131, thus making the starting day the beginning of the year 1929:

```
SUBN(S)      NBR-DAYS, 131; /* Was: Aug 23, 1928, Now: Jan 01, 1929 */
```

Four consecutive years, called a *period*, contains 1461 days. The number of periods that have elapsed and the number of days within the last period are simply the quotient and the remainder of the above NBR-DAYS divided by 1461:

```
DI VREM      NBR-PERIODS, NBR-DAYS, 1461, NBR-DAYS;
```

Now, the number of years elapsed is the number of periods times four; add to that the beginning year, 1929, and we are almost there:

```
MULT         NBR-YEARS, NBR-PERIODS, 4;
ADDN(S)      NBR-YEARS, 1929;
```

The number of days within the last period divided by the number of days in a year gives the number of complete years that must still be added:

```
DI VREM      ADD-YEARS, NBR-DAYS, 365, NBR-DAYS;
ADDN         YEAR, NBR-YEARS, ADD-YEARS;
```

We accumulate the result as *zoned* numbers in:

```
DCL DD YYYYMMDDHHMMSS CHAR(14);
DCL DD YEAR ZND(4,0) DEF(YYYYMMDDHHMMSS) POS( 1);
DCL DD MONTH ZND(2,0) DEF(YYYYMMDDHHMMSS) POS( 5);
DCL DD DAY ZND(2,0) DEF(YYYYMMDDHHMMSS) POS( 7);
DCL DD HOUR ZND(2,0) DEF(YYYYMMDDHHMMSS) POS( 9);
DCL DD MIN ZND(2,0) DEF(YYYYMMDDHHMMSS) POS(11);
DCL DD SEC ZND(2,0) DEF(YYYYMMDDHHMMSS) POS(13);
```

The remainder was the number of days within the last incomplete year. We save that in NBR-OF-DAYS for later on. Because of the choice of starting year, 1929, the last year of any period (for which ADD-YEARS is 3) is a leap year, e.g. 1932. If ADD-YEARS is less than 3, we know that the year is not a leap year. The task now is to find the month based of NBR-DAYS. We define a table with 12 entries where entry number *m* is the number of days from the beginning of the year until month *m*. To find the month, we loop through the table from the end until the entry value is not less than NBR-DAYS:

```
/* Day Base for: JanFebMarAprMayJunJulAugSepOctNovDec*/
DCL DD DAYS CHAR(36) INIT("000031059090120151181212243273304334");
DCL DD DAYS-ACCUM(12) ZND(3,0) DEF(DAYS) POS(1);
```

```
CPY NV      M, 13;
CMP NV(B)   ADD-YEARS, 3/LO(FIND-MONTH);
LEAP-YEAR:
CMP NV(B)   NBR-DAYS, 59/LO(FIND-MONTH), EQ(FEB-29TH);
SUBN(S)     NBR-DAYS, 1;
FIND-MONTH:
SUBN(S)     M, 1;
SUBN(B)     DAY-MONTH, NBR-DAYS, DAYS-ACCUM(M)/NEG(FIND-MONTH);
ADDN        DAY, DAY-MONTH, 1;
CPY NV(B)   MONTH, M/POS(COMPUTE-TIME);
FEB-29TH:
CPY NV      MONTH, 2;
CPY NV      DAY, 29;
```

```
COMPUTE-TIME:
```

Study the above code carefully. Note especially how February 29<sup>th</sup> is handled. Note also the use of *two* branch-conditions in:

```
CMPNV(B)    NBR-DAYS, 59/LO(FIND-MONTH), EQ(FEB-29TH);
```

Because the month value, M, is always positive (that is: greater than zero), we can save a branch-instruction by using POS as a branch-condition in

```
CPYNV(B)    MONTH, M/POS(COMPUTE-TIME);
```

to avoid falling into the FEB-29TH processing. The final step is to compute the hours, minutes, and seconds:

```
COMPUTE-TIME:
DI VREM     HOUR, NBR-SECONDS, 3600, NBR-SECONDS;
DI VREM     MIN,  NBR-SECONDS,  60,  SEC;
```

## Complete Program to Show Current Date and Time

Putting everything together we get the following complete program (**MI RTVDT**):

```
DCL DD MACHINE-CLOCK CHAR(2) INIT(X'0100');
DCL SPCPTR .MACHINE-ATTR INIT(MACHINE-ATTR);
DCL DD     MACHINE-ATTR CHAR(24) BDRY(16);
DCL DD     MAT-MAX-SIZE  BIN(4) DEF(MACHINE-ATTR) POS( 1) INIT(16);
DCL DD     MAT-ACT-SIZE  BIN(4) DEF(MACHINE-ATTR) POS( 5);
DCL DD     MAT-TIMESTAMP CHAR(8) DEF(MACHINE-ATTR) POS( 9);
DCL DD     MAT-TIME-HI  BIN(4) UNSGND DEF(MAT-TIMESTAMP) POS(1);
DCL DD     MAT-TIME-LO  BIN(4) UNSGND DEF(MAT-TIMESTAMP) POS(5);

DCL DD TIMESTAMP      PKD(21,0); /* CAN HOLD 64-BIT UNSIGNED */
DCL DD TIMESTAMP-HI    PKD(11,0);
DCL DD TIMESTAMP-LO    PKD(11,0);
DCL DD TWO**32         PKD(11,0) INIT(P'4294967296');
DCL DD NBR-SECONDS     PKD(15,0);
DCL DD NBR-DAYS        BIN(4);
DCL DD NBR-YEARS       BIN(4);
DCL DD ADD-YEARS       BIN(4);
DCL DD NBR-PERIODS     BIN(4);
DCL DD DAY-MONTH       BIN(4);
DCL DD D               BIN(4);
DCL DD S               BIN(4);
DCL DD M               BIN(4);
DCL DD M               /* DAY BASE FOR: JanFebMarAprMayJunJul AugSepOctNovDec */
DCL DD DAYS CHAR(36) INIT("000031059090120151181212243273304334");
DCL DD DAYS-ACCUM (12)ZND(3,0) DEF(DAYS) POS(1);

DCL DD YYYYMMDDHHMMSS CHAR(14);
DCL DD YEAR  ZND(4,0) DEF(YYYYMMDDHHMMSS) POS( 1);
DCL DD MONTH ZND(2,0) DEF(YYYYMMDDHHMMSS) POS( 5);
DCL DD DAY   ZND(2,0) DEF(YYYYMMDDHHMMSS) POS( 7);
DCL DD HOUR  ZND(2,0) DEF(YYYYMMDDHHMMSS) POS( 9);
DCL DD MIN   ZND(2,0) DEF(YYYYMMDDHHMMSS) POS(11);
DCL DD SEC   ZND(2,0) DEF(YYYYMMDDHHMMSS) POS(13);

/*****

MATMATR .MACHINE-ATTR, MACHINE-CLOCK;
CPYNV   TIMESTAMP-LO, MAT-TIME-LO;
CPYNV   TIMESTAMP-HI, MAT-TIME-HI;
MULT    TIMESTAMP, TIMESTAMP-HI, TWO**32;
ADDN(S) TIMESTAMP, TIMESTAMP-LO;
DIV(SR) TIMESTAMP, 4096000000; /* NOW SECONDS */
ADDN(S) TIMESTAMP, 43386; /* 12:03:06 PM */
DI VREM NBR-DAYS, TIMESTAMP, 86400, NBR-SECONDS;
SUBN(S) NBR-DAYS, 131; /* WAS: AUG 23,1928, NOW: JAN 01,1929 */
DI VREM NBR-PERIODS, NBR-DAYS, 1461, NBR-DAYS; /* 4 YEARS */
MULT    NBR-YEARS, NBR-PERIODS, 4;
ADDN(S) NBR-YEARS, 1929;
DI VREM ADD-YEARS, NBR-DAYS, 365, NBR-DAYS;
ADDN    YEAR, NBR-YEARS, ADD-YEARS;
CPYNV   M, 13;
CMPNV(B) ADD-YEARS, 3/LO(FIND-MONTH);
LEAP-YEAR:
CMPNV(B) NBR-DAYS, 59/LO(FIND-MONTH), EQ(FEB-29TH);
SUBN(S) NBR-DAYS, 1;
FIND-MONTH:
SUBN(S) M, 1;

*****/
```

```

SUBN(B)    DAY-MONTH, NBR-DAYS, DAYS-ACCUM(M)/NEG(FIND-MONTH);
ADDN       DAY, DAY-MONTH, 1;
CPYNV(B)   MONTH, M/POS(COMPUTE-TIME);
FEB-29TH:
CPYNV      MONTH, 2;
CPYNV      DAY, 29;

COMPUTE-TIME:
DI VREM    HOUR, NBR-SECONDS, 3600, NBR-SECONDS;
DI VREM    MIN, NBR-SECONDS, 60, SEC;

SHOW-DATE-TIME:
CPYBLAP    MSG-TEXT, YYYYMMDDHHMMSS, " ";
CALLI      SHOW-MESSAGE, *, .SHOW-MESSAGE;
RTX        *;

%INCLUDE SHOWMSG

```

If we wanted to include the usual date/time separators, we could have defined them as part of the result structure:

```

DCL DD YYYYMMDDHHMMSS CHAR(19);
DCL DD YEAR ZND(4,0) DEF(YYYYMMDDHHMMSS) POS( 1);
DCL DD * CHAR(1) DEF(YYYYMMDDHHMMSS) POS( 5) INIT("/");
DCL DD MONTH ZND(2,0) DEF(YYYYMMDDHHMMSS) POS( 6);
DCL DD * CHAR(1) DEF(YYYYMMDDHHMMSS) POS( 8) INIT("/");
DCL DD DAY ZND(2,0) DEF(YYYYMMDDHHMMSS) POS( 9);
DCL DD * CHAR(1) DEF(YYYYMMDDHHMMSS) POS(11) INIT(" ");
DCL DD HOUR ZND(2,0) DEF(YYYYMMDDHHMMSS) POS(12);
DCL DD * CHAR(1) DEF(YYYYMMDDHHMMSS) POS(14) INIT(":");
DCL DD MIN ZND(2,0) DEF(YYYYMMDDHHMMSS) POS(15);
DCL DD * CHAR(1) DEF(YYYYMMDDHHMMSS) POS(17) INIT(":");
DCL DD SEC ZND(2,0) DEF(YYYYMMDDHHMMSS) POS(18);

```

## Leap Seconds

Civil time is occasionally adjusted by one second increments to ensure that the difference between a uniform time scale defined by atomic clocks does not differ from the Earth's rotational time by more than 0.9 seconds. The Earth is constantly undergoing a deceleration caused by the braking action of the tides. Through the use of ancient observations of eclipses, it is possible to determine the average deceleration of the Earth to be roughly 1.4 milliseconds per day per century. This deceleration causes the Earth's rotational time to slow with respect to the atomic clock time. Modern studies have indicated that the epoch at which the mean solar day was exactly 86,400 seconds was approximately 1820. Before then, the mean solar day was shorter than 86,400 seconds and since then it has been longer than 86,400 seconds.

The length of the mean solar day has increased by roughly 2 milliseconds since it was exactly 86,400 seconds of atomic time about 1.80 centuries ago (i.e. the 180 year difference between 2000 and 1820). That is, the length of the mean solar day is at present about 86,400.002 seconds instead of exactly 86,400 seconds. Over the course of one year, the difference accumulates to almost one second, which is compensated by the insertion of a leap second with a current regularity of a little less than once per year. Other factors also affect the Earth, some in unpredictable ways, so that it is necessary to monitor the Earth's rotation continuously. In order to keep the cumulative difference less than 0.9 seconds, a leap second is added to the atomic time from time to time. This leap second can be either positive or negative depending on the Earth's rotation. Since the first leap second in 1972, all leap seconds have been positive and there were 23 leap seconds in the 29 years to January, 2001.

The decision to insert a leap second is the responsibility of the International Earth Rotation Service in Paris. At the time of writing (February, 2001) the latest information was that no leap second was introduced at the end of December, 2000. Here is a list of all leap seconds:

1972	JAN	1	=JD	2441317.5	TAI -UTC=	10.0
1972	JUL	1	=JD	2441499.5	TAI -UTC=	11.0
1973	JAN	1	=JD	2441683.5	TAI -UTC=	12.0
1974	JAN	1	=JD	2442048.5	TAI -UTC=	13.0
1975	JAN	1	=JD	2442413.5	TAI -UTC=	14.0
1976	JAN	1	=JD	2442778.5	TAI -UTC=	15.0
1977	JAN	1	=JD	2443144.5	TAI -UTC=	16.0
1978	JAN	1	=JD	2443509.5	TAI -UTC=	17.0
1979	JAN	1	=JD	2443874.5	TAI -UTC=	18.0

1980	JAN	1	=JD	2444239.5	TAI -UTC=	19.0
1981	JUL	1	=JD	2444786.5	TAI -UTC=	20.0
1982	JUL	1	=JD	2445151.5	TAI -UTC=	21.0
1983	JUL	1	=JD	2445516.5	TAI -UTC=	22.0
1985	JUL	1	=JD	2446247.5	TAI -UTC=	23.0
1988	JAN	1	=JD	2447161.5	TAI -UTC=	24.0
1990	JAN	1	=JD	2447892.5	TAI -UTC=	25.0
1991	JAN	1	=JD	2448257.5	TAI -UTC=	26.0
1992	JUL	1	=JD	2448804.5	TAI -UTC=	27.0
1993	JUL	1	=JD	2449169.5	TAI -UTC=	28.0
1994	JUL	1	=JD	2449534.5	TAI -UTC=	29.0
1996	JAN	1	=JD	2450083.5	TAI -UTC=	30.0
1997	JUL	1	=JD	2450630.5	TAI -UTC=	31.0
1999	JAN	1	=JD	2451179.5	TAI -UTC=	32.0

Since the internal processor clock drifts more than the Earth's rotation, no leap seconds are used on the AS/400. But to calculate accurate durations you could use the above table.

### ***Higher Resolution in Later Versions***

Later versions of the AS/400 seem to have a better time resolution. As of V4R3 or later, the system clock is 49 bits long (versus 41 bits in earlier versions), but it is returned in a 64-bit unsigned integer as the first (high-order) 49 bits. The low-order 15 bits are not clocked. They are used, though. No two succeeding calls to materializing the clock will return the same number. The low-order 15 bits are incremented to cover the situation of two calls within the same clock tick. There are 8 microseconds to a single clock tick. This is the finest resolution presented by the clock above the MI. This translates to a precision of 0.008 milliseconds.

Blank

## Chapter 5

### Calling Other Programs

#### External Program Objects

The Original Program Model with its run-time binding of programs was once touted as an important advantage of the AS/400 over other platforms. Later the pendulum swung completely to the other extreme where linking modules together was the fashion and late binding was being shunned as being old-fashioned. Then with Java, the pendulum swung back again. Actually, both approaches have their advantages depending on what your requirements are. A program is a separate, directly executable object having (as all objects at the MI-level) both an external *symbolic* representation (CONTEXT, PROGRAM) and an internal representation as a resolved *system pointer*. A program is called using the **CALLX** (Call eXternal) instruction with the first operand being (ordinarily) a system pointer to the program object. A second operand specifies an argument list. A third operand identifies a list of alternate return points, but is usually left as a null operand (“\*”) which will cause the next MI instruction after the CALLX to be executed after the program called by the CALLX returns control.

#### Obtaining a System Pointer

There are several ways of obtaining a system pointer:

- Initializing the pointer using the **INIT** clause on a Declare statement
- Resolving the pointer using the Resolve System Pointer instruction (**RSLVSP**)
- Copying an existing pointer using the Copy Bytes With Pointers instruction (**CPYBWP**)
- Retrieving a pointer from the System Entry Point Table (SEPT - discussed in a later chapter)
- Counterfeiting the pointer (discussed in a later chapter)

#### Initializing a System Pointer

Here is the main syntax for the initial value of a program system pointer:

```
DCL SYSPTR pointername INIT("programname", CTX("libraryname"), TYPE(PGM))
```

The initialization takes place the first time the pointer is used in a session.

#### Resolving a System Pointer

You can also explicitly insert addressability to an object into a system pointer using the Resolve System Pointer instruction (**RSLVSP**). The instruction takes a 34-byte structure (the *template*) as its second operand and returns a resolved system pointer in its first operand (provided that the object can be found):

```
DCL DD RESOLVE CHAR(34);
DCL DD RESOLVE-TYPE CHAR( 2) DEF(RESOLVE) POS( 1);
DCL DD RESOLVE-NAME CHAR(30) DEF(RESOLVE) POS( 3);
DCL DD RESOLVE-AUTH CHAR( 2) DEF(RESOLVE) POS(33);

DCL SYSPTR .MYPGM;

CPYBLA      RESOLVE-TYPE, X' 0201' ;
CPYBLAP     RESOLVE-NAME, "MYPGM", " ";
RSLVSP      .MYPGM, RESOLVE, *, *;
```

The object type (“02” for a program) and subtype (ordinarily “01”) specify what kind of object we are trying to address. A list of a number of AS/400 objects showing their types and subtypes can be found in Appendix B. Note that the object name is 30 characters long padded as necessary on the right with blanks, hence the CPYBLAP-instruction. The last element of the structure (and the last operand) used to contain the authority that should be associated with the pointer. The ability to set authority in a pointer is no longer used for user-domain objects because it constitutes a potential security breach (there is no way to retract or remove the authority once it has been granted). The third operand is a system pointer identifying a context to be searched, or null (“\*”) specifying that the library list should be used:

```
RSLVSP      system pointer, template, context, authority
```

## Copying a System Pointer

In Chapter 3 we saw an example where a stack entry was copied to a local holding area. Because the entry contained a pointer that we needed to use later on, a special copy-instruction “Copy Bytes With Pointers” (**CPYBWP**) was used that maintain the tag bits of the pointer:

```
NEXT-PROGRAM:
  ADDN(S)      PGM-NBR, 1;
  CMPNV(B)     PGM-NBR, STK-NBR-OF-ENTRIES/HI (DONE);
  CPYBWP      THE-ENTRY, STK-ENTRY(PGM-NBR);
```

## System Entry Point Table (SEPT)

Every AS/400 has a space object called the System Entry Point Table, or SEPT for short. The SEPT is a table containing ‘pre-resolved’ system pointers to a large number of OS/400 programs, and the location within the table for a particular system pointer (that is for general use) has remained the same across every OS/400 release. The SEPT allows a quick look-up of the system pointer for a particular system program that can then be used as the second parameter in the CALLX instruction. The SEPT will be covered briefly later in this chapter, and in much more detail in a later chapter.

## Counterfeiting a Pointer

Although not part of the normal programming paradigm for the AS/400 it is possible to construct pointers “out of thin air” for RISC-based AS/400s. This is possible because RISC-based AS/400s rely on *detecting* this blatant security breach instead of *preventing* it. The detection code can be removed and you end up with a valid pointer (valid, but counterfeit – in the sense of not being issued (and checked) by the system). We shall also discuss this in detail, in later chapters.

## Encapsulated Objects

With a system pointer to an object you can perform operations on the object that are appropriate for the object: you can execute a program, search an index, retrieve controlled information about the object, etc. You are generally *not supposed* to be able to access the object in an uncontrolled manner, such as inspecting or changing data internal to the object. Objects are said to be encapsulated and thus inviolate. To access or change data inside an object you need a *space pointer* to that data. There is an MI-instruction to create a space pointer from another pointer, “Set Space Pointer From Pointer”, **SETSPFP**, so you might think that the following code fragment would do the trick:

```
DCL SYSPTR .SYSPTR; /* a system pointer */
DCL SPCPTR .SPCPTR; /* a space pointer */

SETSPFP .SPCPTR, .SYSPTR; /* set space pointer */
```

In fact, it does create a valid space pointer, but *not* to the system object as we want. Instead, a space pointer to the “primary associated space” is created. For many encapsulated objects – especially program objects, the associated space does not contain much of interest, so the freshly minted space pointer is often rather useless. This is, of course, carefully designed to be so.

## The Argument List

This is also called the *operand list*, hence the “**OL**” in the syntax for declaring it. First, here is how you declare the argument list in the *calling* program (the caller):

```
DCL OL name (parg1, parg2, ..., pargn) ARG
```

Second, here is how you declare the corresponding list in the *called* program (the callee):

```
DCL OL name (parg1, parg2, ..., pargn) PARM EXT MIN(m)
```

The number, *m*, of arguments passed (or parameters received) can be smaller (**MIN(m)**) than the maximum number, *n*, implied by the number of items (*pargn*) in the list. Up to 255 items can be specified. The “Store

Parameter List Length"-instruction (**STPLLEN**) can be used to obtain the actual number of parameters passed.

The arguments (parameters) are almost always *space pointers* to data-items, although they don't *have* to be. Many HLLs require arguments to be space pointers, so if you wish to be sure that your program could be called by an HLL-program it is wise to stick to that convention. The arguments must be declared *before* the operand list.

### Program Call Example

To illustrate the technique we show two sample programs. **MI CPGM1** calls **MI CPGM2** with two arguments. **MI CPGM2** can handle up to three parameters and simply replaces the values of any parameters received by a suitable text, which is displayed by **MI CPGM1** when control returns to it.

Here is the main program:

```
DCL SPCPTR . ARG1 INIT(ARG1); /* pointer to 1st argument */
DCL DD     ARG1 CHAR(10);    /* 1st argument */

DCL SPCPTR . ARG2 INIT(ARG2); /* pointer to 2nd argument */
DCL DD     ARG2 CHAR(10);    /* 2nd argument */

DCL OL      MI CPGM2 (. ARG1, . ARG2) ARG; /* argument list */
DCL SYSPTR . MI CPGM2;      /* system pointer to callee */

DCL DD RESOLVE CHAR(34);
DCL DD RESOLVE-TYPE CHAR( 2) DEF(RESOLVE) POS( 1);
DCL DD RESOLVE-NAME CHAR(30) DEF(RESOLVE) POS( 3);
DCL DD RESOLVE-AUTH CHAR( 2) DEF(RESOLVE) POS(33);

ENTRY * EXT;
RESOLVE-TO-PGM:
  CPYBLA    RESOLVE-TYPE, X'0201'; /* program type */
  CPYBLAP   RESOLVE-NAME, "MI CPGM2", " "; /* 30-char name */
  RSLVSP    . MI CPGM2, RESOLVE, *, *; /* * is liblist */

LOAD-ARGUMENTS-AND-CALL:
  CPYBLAP   ARG1, "ARG1", " "; /* value of 1st arg */
  CPYBLAP   ARG2, "ARG2", " "; /* value of 2nd arg */

  CALLX     . MI CPGM2, MI CPGM2, *; /* call the program */

BACK-FROM-CALL:
  CAT      MSG-TEXT, ARG1, ARG2; /* concatenate ARGs */
  CALLI     SHOW-MESSAGE, *, . SHOW-MESSAGE;

RETURN:
  RTX      *;

%I NCLUDE SHOWMSG
```

Note the **ENTRY** statement. It defines an entry point to the program. By default that would be the first executable instruction, but you can also define it explicitly as we just did. In our example, it has no special name ("\*") and is an *external* entry point (**EXT**). Note also the use of the Concatenate-instruction (**CAT**) that concatenates ARG1 and ARG2 into MSG-TEXT, padding with blanks on the right.

The called program, **MI CPGM2**, accepts up to three parameters. Each parameter is a space pointer declared to be a parameter (**PARM**):

```
DCL SPCPTR . PARM1 PARM;
DCL DD     PARM1 CHAR(10) BAS(. PARM1);
```

The PARM1 data item is *based* on a parameter space pointer (. PARM1) as indicated by the BAS clause.

```
DCL SPCPTR . PARM2 PARM;
DCL DD     PARM2 CHAR(10) BAS(. PARM2);

DCL SPCPTR . PARM3 PARM;
DCL DD     PARM3 CHAR(10) BAS(. PARM3);
```

The parameters are external (**EXT**) and at least one must be present:



```
DCL OL MI CPGM2 (. PARM1, . PARM2, . PARM3) PARM EXT MIN(1);
DCL DD NBR-PARMS BIN(2);
```

The entry point specifies the operand list defining possible parameters:

```
ENTRY * (MI CPGM2) EXT;
      STPLEN  NBR-PARMS; /* get number of actual parameters */
      CPYBLAP PARM1, "PARM1", " "; /* just mark the parameter */
```

We could unconditionally return “PARM1” in the first parameter because we are assured by the system that there will be at least one. For the remaining parameters, we need a guard based on the number of parameters to only allow returning a value if the parameter is present:

```
CMPNV(B)  NBR-PARMS, 2/LO(=+2); /* if NBR-PARMS not < 2 */
CPYBLAP   PARM2, "PARM2", " "; /*   parm2 = "PARM2" */

CMPNV(B)  NBR-PARMS, 3/LO(=+2); /* if NBR-PARMS not < 3 */
CPYBLAP   PARM3, "PARM3", " "; /*   parm3 = "PARM3" */
```

At the end, we simply return to the caller:

```
RETURN:
      RTX      *;
```

## Relative Branch Conditions

Did you notice the curious branch targets (=**+2**) in the above guards? Let me rewrite the guards in a more traditional way:

```
CMPNV(B)  NBR-PARMS, 2/LO(AFTER-2);
CPYBLAP   PARM2, "PARM2", " ";
AFTER-2:
CMPNV(B)  NBR-PARMS, 3/LO(AFTER-3);
CPYBLAP   PARM3, "PARM3", " ";
AFTER-3:
```

Labels should be informative as to what the code is doing and not simply place names. I had to invent two labels with little informational contents (AFTER-2 and AFTER-3) just to have targets for the branch conditions. Instead of using an actual label name as a branch condition, MI allows *relative* branch targets. Both positive and negative values may be used. The relative condition ‘=**+2**’ means skip forward two (2) instructions (*including* the instruction with this condition), so we could just as well have written:

```
CMPNV(B)  NBR-PARMS, 2/LO(=+2);
CPYBLAP   PARM2, "PARM2", " ";
AFTER-2:
CMPNV(B)  NBR-PARMS, 3/LO(=+2);
CPYBLAP   PARM3, "PARM3", " ";
AFTER-3:
```

As always, you can only jump to a branch point (the ‘:’ after the label). Because MI is free-form, we can write the label on the same line as the instruction being skipped:

```
CMPNV(B)  NBR-PARMS, 2/LO(=+2);
CPYBLAP   PARM2, "PARM2", " "; AFTER-2:

CMPNV(B)  NBR-PARMS, 3/LO(=+2);
CPYBLAP   PARM3, "PARM3", " "; AFTER-3:
```

Finally, since we are actually not using the label names, we can simply omit them (but keep the colons):

```
CMPNV(B)  NBR-PARMS, 2/LO(=+2);
CPYBLAP   PARM2, "PARM2", " "; :

CMPNV(B)  NBR-PARMS, 3/LO(=+2);
CPYBLAP   PARM3, "PARM3", " "; : :
```

Now the code is no longer cluttered with the arbitrary, invented labels. Some people indent the guarded instructions for added clarity (?):

```
CMPNV(B)  NBR-PARMS, 2/LO(=+2);
      CPYBLAP   PARM2, "PARM2", " "; :
```

```

CMPNV(B)      NBR-PARMS, 3/LO(=+2);
CPYBLAP      PARM3, "PARM3", " ";

```

## An Optimization

Since resolving a pointer takes time, one often uses a first-time flag to avoid unnecessary, subsequent executions of the resolve instruction. The RESOLVE-TYPE can serve as a convenient first-time flag having an initial value of all zeroes:

```

DCL DD RESOLVE CHAR(34);
DCL DD RESOLVE-TYPE CHAR( 2) DEF(RESOLVE) POS( 1) INIT(X'0000');
DCL DD RESOLVE-NAME CHAR(30) DEF(RESOLVE) POS( 3);
DCL DD RESOLVE-AUTH CHAR( 2) DEF(RESOLVE) POS(33);

CMPBLA(B)      RESOLVE-TYPE, X'0000' /NEQ(=+4); /* if TYPE = zeroes */
CPYBLA        RESOLVE-TYPE, X'0201';           /* TYPE = 0201 */
CPYBLAP        RESOLVE-NAME, "MYPGM", " ";      /* NAME = ... */
RSLVSP        .MYPGM, RESOLVE, *, *;           /* Resolve .MYPGM */

```

## The System Entry Point Table

To speed up access to operating system functions and also to prevent programs with the same name being present in a context (library) on the library list to be called instead, as you remember, the system maintains a list of *pre-resolved* system pointers to operating systems functions and APIs called the System Entry Point Table (the SEPT). It contains more than 6,000 entries and the number grows with each release of OS/400. In a later chapter we shall look at the SEPT in detail. For now, we'll just touch briefly upon how to use the SEPT. We have used the Send Message API, **QMHSNDM**, several times (in the SHOWMSG include file). It is SEPT entry number 4268. We call the API like this:

```

CALLX      .SEPT(4268), QMHSNDM, *; /* SEND MSG TO MSGQ */

```

The SEPT is accessed through a pointer found in the *Process Communication Object* (about which much more later). Here is how we declare the array of system pointers, which is the SEPT:

```

DCL SYSPT .SEPT(6440) BAS(SEPT-POINTER);
DCL SPC PROCESS-COMMUNICATION-OBJECT BASPCO;
DCL SPCPTR SEPT-POINTER DIR;

```

The secret to all this is the curious **BASPCO** clause, meaning “based on the PCO (Process Communication Object)”. A list of public entries into the SEPT and their numbers will be given in appendix D.

Blank

## Chapter 6

### RISC Code for Setting a Pointer

#### ***Set Space Pointer From Pointer***

To learn about how pointers are treated below the MI level, i.e. at the RISC-code level, let's compile the following simple MI-program (**MI STPTR1**):

```
DCL DD POINTERS CHAR(32) BDRY(16);
DCL SYSPTR .SYSPTR DEF(POINTERS) POS( 1);
DCL SPCPTR .SPCPTR DEF(POINTERS) POS(17);

SETSPFPF .SPCPTR, .SYSPTR; /* Set space pointer from pointer */
```

The program attempts to set a space pointer from a system pointer (albeit uninitialized). We shall now investigate the RISC code generated by the MI-compiler. With that, we shall start the process of learning both about the RISC PowerPC machine architecture and about the use of it by the AS/400. The going may be rough in places, but this is complicated stuff and there is no other way than to dig into it. You may want to just skim this chapter to get a feeling for what we have here, and the return to it later.

#### **Documentation About the PowerPC**

For some information about the PowerPC that is used in the RISC AS/400 models, see Motorola's technical specifications online at <http://www.mot.com/SPS/PowerPC/teksupport/teklibrary/manuals/PRG.pdf> for a quick reference guide. The full programming environment manual is 828 pages long and can be found at this link <http://www.motorola.com/SPS/PowerPC/teksupport/teklibrary/manuals/pem64b.pdf>. To read PDF files you will need Adobe's Acrobat reader at <http://www.adobe.com>. There is also some older version of this manual at <http://www-3.ibm.com/chips/techlib/techlib.nsf/productfamilies/PowerPC>. All of these links are subject to change.

These documents do not cover the extensions that IBM has added to support the AS/400. There is no public documentation about the AS/400 extensions which cover about two dozen new instructions. The present book you are reading might be the only place where you'll find such information.

#### ***AS/400 RISC PowerPC Machine Architecture***

It is hard to boil the 828 pages of the manual down to a few paragraphs, but luckily we don't need all the gory details. Basic data types are (8-bit) bytes, 2-byte halfwords, 4-byte words, 8-byte doublewords, and 16-byte "quadwords". All of these should be aligned on appropriate boundaries for best performance. Quadwords must always be aligned. A general "string" type with no alignment requirement is also supported. The machine has 32 general-purpose 64-bit registers, 32 floating-point 64-bit registers and a number of special purpose registers of various sizes. Instructions are always 4 bytes long and are aligned on word boundaries. In spite of the RISC appellation (*i.e.* of a *Reduced* Instruction Set Computer) there is a multitude of different instruction formats. At last count, 15 main formats encompassing 61 variations. A different classification (but a useful one) is based on the number of operands used in each instruction. The number of operands ("addresses") ranges from zero to three.

Three-address instructions are typically register only operations in keeping with classical RISC ideas, e.g.

ADD Rd, Ra, Rb ; Rd = Ra + Rb, the sum of Ra and Rb is placed in Rd

Many two-address instructions are load/store instructions:

LD Rd, D(Ra) ; Rd = doubleword at address (Ra) + Displacement

Branch instructions (GOTOs) are one-address instructions. Either to an *absolute* address as in:

BA AA ; Goto absolute address AA

Or to a relative address:

B +RA ; Goto instruction at address RA higher than address of instruction

Typically, you load several registers, operate on them, and finally store the result. Nothing new here. We shall introduce and explain instructions as needed.

## System Service Tools

For serious Machine-Level programming you need access to the *System Service Tools* (SST). A special authority (\*SERVICE) is required for that. Even if you do not have access to SST, you should still follow along to get an appreciation of the inner workings of the AS/400. You start SST with the **STRSST** command. Below we show screenshots of the selection process (very ‘pedantic’). On V4R5M0 and above, the references to ‘diskette’ in some of the menu choices have been removed from the screens, but the basic navigation through the screens, as shown below, will be the same. Select the highlighted choices:

### System Service Tools (SST)

Select one of the following:

1. Start a service tool
2. Work with active service tools
3. Work with disk units
4. Work with diskette data recovery
5. Work with system partitions

### Start a Service Tool

Warning: Incorrect use of this service tool can cause damage to data in this system. Contact your service representative for assistance.

Select one of the following:

1. Product activity log
2. Trace Licensed Internal Code
3. Work with communications trace
4. Display/Alter/Dump
5. Licensed Internal Code log
6. Main storage dump manager
7. Hardware service manager

### Display/Alter/Dump Output Device

Select one of the following:

1. Display/Alter storage
2. Dump to printer
3. Dump to diskette
4. Dump to tape
5. Print diskette dump file
6. Print tape dump file
7. Display dump status

### Select Data

Select one of the following:

1. Machine Interface (MI) object
2. Licensed Internal Code (LIC) data
3. LIC module
4. Tasks/Processes
5. Starting address

### Select MI Object

Select one of the following:

1. Access group (01)
2. Program (02)
3. Module (03)
4. Permanent context (04)
5. Temporary context (04)
6. Byte string space (06)
7. Journal space (07)
8. User profile (08)
9. Journal port (09)
10. Queue (0A)

## Find MI Object

Select one of the following:

1. Find by object name and context name
2. Find by object address

### Find By Object Name And Context Name

Type choices, press Enter.

```
Object:
Type . . . . . : (02) - Program
Name . . . . . : MI STPTR1
Subtype . . . . . : 01 00-FF
```

```
Context:
Name . . . . . : YOURLIB
Subtype . . . . . : 01 00-FF
```

### Display Object Found Information

```
Object:
Type . . . . . : (02) - Program
Name . . . . . : MI STPTR1
Subtype . . . . . : 01
```

```
Context:
Name . . . . . : YOURLIB
Subtype . . . . . : 01
```

```
User profile:
Name . . . . . : YOURUPRF
Subtype . . . . . : 01
```

### Select Format

Select one of the following:

1. Dump in hexadecimal
2. Dump in hexadecimal (logical blocks)
3. Format dis-assembled code

On each screen you select the highlighted option. The dump is spooled to file QPCSMPT and can be examined with the **WRKSPLF** command:

### Work with All Spooled Files

Opt	File	User	Device or Queue	User Data	Sts	Total Pages	Cur Page	Copy
—	<b>QPCSMPT</b>	YOURUPRF	YOURQUEUE		RDY	23		1

Way down on page 14 of the listing we find the RISC code for our program. One important column of the listing which will prove to be an invaluable help to us, is the 'MI INSTR NBR' column. This column maps the MI instruction number found in the MI-compiler output spoolfile when you compile an MI source listing, with a particular RISC assembly instruction and with this information, you can easily determine the 'blocks' of RISC instructions that map to parts of your code. I have edited the listing to fit and added a few annotations:

```

      DISPLAY/ALTER/DUMP
MI PROGRAM SUBTYPE: 01 NAME: MI STPTR1 ADDRESS: 2221D0DDE4 000000
RISC INSTRUCTIONS (MI STPTR1)
      ADDRESS LOCATION OBJECT TEXT SOURCE STATEMENT MI INSTR NBR
***** ENTRY POINT *****
2221D0DDE4 001480 000000 FB81FF43 STMD 28,0XFF40(1) Initialization code
2221D0DDE4 001484 000004 605C0000 ORI 28,2,0
2221D0DDE4 001488 000008 7C0802A6 MFSPR 0,8
2221D0DDE4 00148C 00000C F8010028 STD 0,0X28(1)
2221D0DDE4 001490 000010 F821FE81 STDU 1,0XFE80(1)
2221D0DDE4 001494 000014 39800001 ADDI 12,0,1
2221D0DDE4 001498 000018 F9810060 STD 12,0X60(1)
2221D0DDE4 00149C 00001C F8410020 STD 2,0X20(1)
2221D0DDE4 0014A0 000020 3C005415 ADDIS 0,0,21525
2221D0DDE4 0014A4 000024 F8010008 STD 0,0X8(1)
2221D0DDE4 0014A8 000028 F82101B8 STD 1,0X1B8(1)
2221D0DDE4 0014AC 00002C 33FF0060 ADDIC 31,31,96

```

```

2221D0DDE4 0014B0 000030 7C2001C8 TXER 1, 0, 35
2221D0DDE4 0014B4 000034 419A8053 BCLA 12, 26, 0X8050
2221D0DDE4 0014B8 000038 419D8183 BCLA 12, 29, 0X8180
2221D0DDE4 0014BC 00003C 607D0000 ORI 29, 3, 0
2221D0DDE4 0014C0 000040 EBD00020 LD 30, 0X20(28)
2221D0DDE4 0014C4 000044 38600082 ADDI 3, 0, 130
2221D0DDE4 0014C8 000048 986100A8 STB 3, 0XA8(1)
***** EXCEPTION/EXIT HANDLERS ENABLED: 1

2221D0DDE4 0014CC 00004C E0DE004B LQ 6, 0X40(30), 11 01 SETSPFPF
2221D0DDE4 0014D0 000050 7C0004C8 TXER 0, 0, 41
2221D0DDE4 0014D4 000054 60C30000 ORI 3, 6, 0
2221D0DDE4 0014D8 000058 38870000 ADDI 4, 7, 0
2221D0DDE4 0014DC 00005C 78C717A1 RLDI CL. 7, 6, 2, 62
2221D0DDE4 0014E0 000060 4082000C BC 4, 2, 0XC
2221D0DDE4 0014E4 000064 4B801963 BLA 0X3801960
2221D0DDE4 0014E8 000068 38830000 ADDI 4, 3, 0
2221D0DDE4 0014EC 00006C E8C08110 LD 6, 0X8110(0)
2221D0DDE4 0014F0 000070 788701E5 RLDI CR. 7, 4, 0, 39
2221D0DDE4 0014F4 000074 7C200348 TXER 1, 0, 38
2221D0DDE4 0014F8 000078 7C0103E6 SETTAG
2221D0DDE4 0014FC 00007C 38E40000 ADDI 7, 4, 0
2221D0DDE4 001500 000080 F8DE0052 STQ 6, 0X50(30)

2221D0DDE4 001504 000084 80C100F8 LWZ 6, 0XF8(1) 02 RTX (i m p l i e d)
2221D0DDE4 001508 000088 78C4FFA3 RLDI CL. 4, 6, 63, 62
2221D0DDE4 00150C 00008C 7C000348 TXER 0, 0, 38
2221D0DDE4 001510 000090 419D81C3 BCLA 12, 29, 0X81C0
2221D0DDE4 001514 000094 419A8063 BCLA 12, 26, 0X8060
2221D0DDE4 001518 000098 38210180 ADDI 1, 1, 384
2221D0DDE4 00151C 00009C E8010028 LD 0, 0X28(1)
2221D0DDE4 001520 0000A0 7C0803A6 MTSR 8, 0
2221D0DDE4 001524 0000A4 EB81FF43 LMD 28, 0XFF40(1)
2221D0DDE4 001528 0000A8 4E800021 BCLRL 20, 0

```

## Dissecting SETSPFPF


We shall now proceed to understand the **SETSPFPF** RISC code. First, let me focus in on the relevant instructions:

```

2221D0DDE4 0014CC 00004C E0DE004B LQ 6, 0X40(30), 11 01 SETSPFPF
2221D0DDE4 0014D0 000050 7C0004C8 TXER 0, 0, 41
2221D0DDE4 0014D4 000054 60C30000 ORI 3, 6, 0
2221D0DDE4 0014D8 000058 38870000 ADDI 4, 7, 0
2221D0DDE4 0014DC 00005C 78C717A1 RLDI CL. 7, 6, 2, 62
2221D0DDE4 0014E0 000060 4082000C BC 4, 2, 0XC
2221D0DDE4 0014E4 000064 4B801963 BLA 0X3801960
2221D0DDE4 0014E8 000068 38830000 ADDI 4, 3, 0

2221D0DDE4 0014EC 00006C E8C08110 LD 6, 0X8110(0)
2221D0DDE4 0014F0 000070 788701E5 RLDI CR. 7, 4, 0, 39
2221D0DDE4 0014F4 000074 7C200348 TXER 1, 0, 38
2221D0DDE4 0014F8 000078 7C0103E6 SETTAG
2221D0DDE4 0014FC 00007C 38E40000 ADDI 7, 4, 0
2221D0DDE4 001500 000080 F8DE0052 STQ 6, 0X50(30)

```



As no assembler is publicly available for the RISC processor (certainly not involving the AS/400 specific extensions; maybe I should write one...) you will have to become fluent in bit manipulation yourself. This is not hard, just a bit tedious (pun intended). The code for **SETSPFPF** already incorporates several AS/400 extensions. Indeed, the very first instruction, **LQ**, as a matter of fact, is one of them: Load Quadword. In dissecting instructions we shall proceed as in the following example:

**E0DE004B** LQ 6, 0X40(30), 11

```

  E   O   D   E   O   O   4   B
1110 0000 1101 1110 0000 0000 0100 1011 ; each hex digit expanded into bits
111000 00110 11110 000000000100 1011 ; bits grouped into instruction fields
LQ=56   R6   R30           4 11 ; R6-R7 = Quadword at (R30) + 4*0X10

```

## LQ - Load Quadword Instruction and Pointer Tag Bits

Every 16-byte “quadword” has a logical *tag*-bit. A *pointer* in the original S/38 occupied four consecutive 4-byte words in memory, each with its own tag bit. The original design was to have the tag bit for every memory word set to 1 if that word contained any of the four parts of a pointer. If there was no part of a

pointer in the word, the tag bit was set to 0. The pointer itself needed only one tag, so we said if all 4 bits in the consecutive four memory words were set to 1, then the pointer had a logical tag of 1. If any of the 4 bits was 0, then the pointer had a logical tag of 0.

Later implementations of the AS/400 have a 64-bit (8-byte) memory word. A 64-bit memory word requires 8 ECC (Error Correction Code) bits; so with the tag bit, AS/400 memories are packaged 73 bits wide. The RISC-versions of the AS/400 still keep pointers on 16-byte boundaries and each word of the pointer has one logical tag bit. For the AS/400 with the 64-bit memory word, the *two* tag bits in the consecutive two words that hold the pointer must both be 1 for the pointer to have a logical tag of 1. If either tag bit is 0, then the pointer has a logical tag of 0. Whenever a write to memory occurs, the memory control hardware creates the ECC and stores it with the memory word. As part of this write operation, the control hardware also turns off the tag bit in the memory word (sets it to 0). Any standard instruction that writes to memory will always result in the tag bits for the words written being set to 0. A pointer is invalid if its logical tag bit is not set.

The Load Quadword (**LQ**) instruction loads 16 bytes from memory into two *consecutive* 64-bit general-purpose registers and sets a bit in a control register if both tag bits in the memory words fetched are 1; otherwise, the bit is set to 0. A complementary instruction, Store Quadword (**STQ**), stores 16 bytes from two consecutive registers into memory and turns on the two tag bits, *if* the tag bit in the control register is still set. (There is also an instruction – **SETTAG** – to set the tag bit in the control register.) The LQ-instruction has a somewhat mysterious format: “**LQ** Rd, offset (Ra), nn”, where “nn” is a 4-bit number that I have seen be 2, 11, or 15. Since a quadword is always aligned on a 16-byte boundary the low-order 4 bits of the address are always zeroes, so need not be present in the offset (a.k.a. the displacement). There is a rumor that the four bits in the “nn” nibble denote an ‘Ored’ mask of pointer types:

B' 1000' = system pointer  
 B' 0100' = instruction pointer  
 B' 0010' = space pointer  
 B' 0001' = data pointer

The generated code for the **CPYBWP** MI-instruction has an **LQ**-instruction with nn = B' 1111' hinting at some truth to the rumor. It is not clear what this is used for.

So back to the code, we now have Register 30 pointing to our local storage area (simply because it is used as such) and **.SYSPTR** being stored at offset hex 40 (or 0X40). Registers 6 and 7 contain the value of the pointer. The following instruction (**TXER**) tests to see if the tag bits were set (as they should be for the pointer to be valid). Not surprisingly, since it deals with tag bits, **TXER** is also an undocumented AS/400 extension:

```
7C0004C8      TXER 0,0,41      ; trap 41 if tag not set

  7      C      0      0      0      4      C      8
0111 1100 0000 0000 0000 0100 1100 1000      ; each hex digit expanded into bits
011111 00000 00000 00000 1001 100100 0      ; bits grouped into fields
  31      0      0      9      36 0      ; X-form trap 41-32 = 9
```

## TXER Instruction

The exact working of this instruction is unclear. Empirically we find that if the tag bits were not set, this instruction traps (i.e. invokes a system exception handler). Presumably, it tests bits in the Integer Exception Register (set by the LQ instruction). Here is a decoding of the other TXER instructions:

```
7C200348      TXER 1,0,38
0111 1100 0010 0000 0000 0011 0100 1000
011111 00001 00000 00000 0110 100100 0
  31      1      0      6      36 0      ; X-form trap 38-32 = 6

7C2001C8      TXER 1,0,35
0111 1100 0010 0000 0000 0001 1100 1000
011111 00001 00000 00000 0011 100100 0
  31      1      0      3      36 0      ; X-form trap 35-32 = 3
```



What *is* significant is that the TXER 0, 0, 41 instruction is the system's *only* guard against a counterfeit pointer. If you replace this instruction with a No Operation (NOP, hex 60000000) a counterfeit pointer is accepted.

The code so far has loaded the pointer, checked the tag bits and is happy that we now have a valid pointer (at least the tag bits are set). The next instruction just copies register 6 to register 3:

```
60C30000    ORI 3, 6, 0    ; ORI Ra, Rs, unsigned immediate operand
6 0 C 3 0 0 0 0
0110 0000 1100 0011 0000 0000 0000 0000 ; each hex digit expanded into bits
011000 00110 00011 000000000000000000 ; bits grouped into instruction fields
ORI=24    R6    R3    0    ; R3 = R6 or immediate value 00...00
```

## ORI - OR Immediate Instruction

The 16-bit *unsigned* immediate address is extended on the left with 48 binary zeroes and is ORed with the source register (Rs). The result is placed in the destination register (Ra). Note that the registers appear reversed in the symbolic representation of the instruction. As used above, the effect is simply to copy R6 to R3. Remember that R6 contains the high-order 64 bits of our, effectively, 128 bit system pointer.

So, we now have a copy of the upper bits of the pointer in R3. The next instruction just copies the low-order bits in R7 to R4:

```
38870000    ADDI 4, 7, 0    ; ADDI Rd, Ra, signed immediate operand
3 8 8 7 0 0 0 0
0011 1000 1000 0111 0000 0000 0000 0000 ; each hex digit expanded into bits
001110 00100 00111 000000000000000000 ; bits grouped into instruction fields
ADDI=14    R4    R7    0    ; R4 = R7 + immediate value 00...00
```

## ADDI - Add Immediate Instruction

The 16-bit *signed* immediate address is extended on the left with 48 copies of the sign bit and is added to Ra. The result is placed in the destination register (Rd). Note that the registers do *not* appear reversed in the symbolic representation of the instruction as they did for the ORI instruction (yet another example from the confusing multitude of instruction formats). As used above, the effect is simply to copy R7 to R4. Remember that R7 contains the low-order 64 bits of our system pointer. An extra little quirk: if the Ra *field* (not the contents of the register Ra) is zero, the contents of the signed immediate operand is simply placed in Rd, *not* added to R0.

So, we now have a copy of the lower bits of the pointer in R4. The next instruction is very powerful, but complicated:

```
78C717A1    RLDI CL. 7, 6, 2, 62 ; Rotate Left Ra, Rs, SH, MB, then Clear on the Left
7 8 C 7 1 7 A 1
0111 1000 1100 0111 0001 0111 1010 0001 ; each hex digit expanded into bits
011110 00110 00111 00010 111101 000 0 1 ; bits grouped into instruction fields
30    R6    R7    *2    *62    0 0 1 ; R7 = R6 rotl 2, clear 62 bits, record
```

## RLDICL Instruction

The “Rotate Left Double Word Immediate then Clear Left (recording)”-instruction:

```
RLDI CL. Ra, Rs, SH, MB (Rightmost bit = 0)
RLDI CL. Ra, Rs, SH, MB (Rightmost bit = 1)
```

works as follows: the contents of Rs are rotated left the number of bits specified by operand SH. Rotated means that bits shifted out at one end are inserted at the other end. A mask is generated having 1 bits from bit MB through bit 63 and 0 bits elsewhere. The rotated data is ANDed with the generated mask and the

result is placed into Ra. Note that both SH and MB are *split fields*. Here the PowerPc betrays its 32-bit roots. Originally, the SB and MB fields were only 5 bits long (because registers were only 32 bits long). To make the fields 6 bits (needed for 64-bit registers) bits from elsewhere in the instruction were pressed into service. The SH field is now constructed by concatenating bit 30 with bits 16 through 20 (bits are numbered from zero and up starting from the left), thus SH = 000010 = 2. The MB field is constructed by concatenating bit 26 with bits 21 through 25, thus MB = 111110 = 62.

Note that RLDI CL can be used to extract, rotate, shift, and clear bit fields using the methods shown below:

- To extract an  $n$ -bit field, that starts at bit position  $b$  in Rs, right-justified into Ra (clearing the remaining  $64 - n$  bits of Ra), set SH =  $b + n$  and MB =  $64 - n$ .
- To rotate the contents of a register left by  $n$  bits, set SH =  $n$  and MB = 0; to rotate the contents of a register right by  $n$  bits, set SH =  $(64 - n)$ , and MB = 0.
- To shift the contents of a register right by  $n$  bits, set SH =  $64 - n$  and MB =  $n$ .
- To clear the high-order  $n$  bits of a register, set SH = 0 and MB =  $n$ .

The effect of RLDI CL. 7, 6, 2, 62 is then first to rotate the two top bits of R6 into the two low-order bits, then clearing all the other bits, effectively extracting the original two top bit as a 64-bit integer in R7. The period after the operation code is the “record result condition” mark causing the CF0 field in the *condition register* to be set according to the resulting value.

## Condition Register

The condition register (CR) is a 32-bit register that reflects the result of certain operations and provides a mechanism for testing and branching. The bits in the CR are grouped into eight 4-bit fields, CR0–CR7. For integer instructions, CR0 bits 0–3 are set to reflect the result as a signed quantity. The CR bits are

- |   |               |  |
|---|---------------|--|
| 0 | Negative (LT) | — This bit is set when the result is negative (less than zero).    |
| 1 | Positive (GT) | — This bit is set when the result is positive (greater than zero). |
| 2 | Zero (EQ)     | — This bit is set when the result is zero.                         |
| 3 | Overflow (SO) | — This bit is set if the result is too large.                      |

With the condition register set, we can now branch depending on the result:

**4082000C** BC 4, 2, 0XC ; branch ahead 12 bytes if not equal zero

<b>4</b>	<b>0</b>	<b>8</b>	<b>2</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>C</b>	
0100	0000	1000	0010	0000	0000	0000	1100	; each hex digit expanded into bits
010000	00100	00010	000000000000	1100				; bits grouped into instruction fields
BC=16	BO	BI		12				; branch ahead 12 bytes if <i>not</i> = zero

## BC - Branch Conditional Instruction

The simple conditional Branch instruction has the format:

BC BO, BI, target (the target address is *relative* to the address of the instruction)

The BI field specifies the bit in the condition register (CR) to be used as the condition of the branch. The BO field is (partly) encoded as follows:

- |       |  |
|-------|--|
| 001zy | Branch if the condition is FALSE (normally 00100 = 4)  |
| 011zy | Branch if the condition is TRUE. (normally 01100 = 12) |
| 1z1zz | Branch always (normally 10100 = 20)                    |

The ‘z’ indicates a bit that is ignored. The ‘y’ bit provides a hint about whether a conditional branch is likely to be taken, and may be used to improve performance. Our branch instruction (BC 4, 2, 0X12) can now be read as: *if* it is FALSE that the current result is EQUAL to zero, *then* branch ahead 12 bytes from the current instruction address. As we shall see, the top two bits of a system pointer are zeroes, so we can restate the situation like this: if the pointer were a system pointer, execute the following two instructions.

```

4B801963    BLA 0X3801960    ; Branch with Link to Absolute address
38830000    ADDI 4, 3, 0

 4   B   8   0   1   9   6   3
0100 1011 1000 0000 0001 1001 0110 0011    ; each hex digit expanded into bits
010010 1110000000000011001011000 1 1      ; bits grouped into instruction fields
B=18                                LI L A    ; Branch to Absolute address with Link

```

## BLA - Branch with Link to Absolute Address Instruction

The branch target address is the signed LI-value padded with two zero bits on the right and then sign-extended on the left to 64 bits. The effective address of the instruction following the branch instruction is placed into the *link register*. The target address for our sample instruction turns out to be FFFFFFFF 801960. There is only room in the instruction for two bits of the ‘F’ just before the ‘801960’, so they appear as ‘3’ in BLA 0X3801960. As the address in the instruction is extended on the left with either zeroes or ones depending on the value of the highest order bit, it is clear that the absolute address will have to be either at the lowest end of the total address range or at the very highest end.

## Link Register

The link register contains the return address for a subroutine. The “Branch Conditional to Link Register” instruction (BCLR) can be used to return to the instruction following the subroutine call. Note that the machine does not use a hardware call stack.

## Addresses, Segments, Offsets, and SLIC

A 64-bit (8-byte) *Virtual Address* is composed of a 40-bit (5-byte) *Segment* value and a 24-bit (3-byte) *Offset* value. Virtual addresses are treated as unsigned quantities and are mapped to real storage addresses through the memory management unit of the PowerPC. The System Internal Licensed Code (SLIC) that makes up OS/400 below the MI-level is mostly found at the high end of the virtual address range.

The branch to segment (FFFFFFFF) offset (801960) is then really a call of a SLIC routine. In fact, to an address in a *table* at entry point .bl a\_md1\_NORTHSTAR<sup>1</sup> in module #cfnsbl a, from where the machine instantly jumps (with a ‘Branch to Absolute address’ instruction, BA) to FFFFFFFF 70D2E0, entry point “.setspaceptrfromptr” in module “#oxsppfp”). Since the link register was not specified in the second branch instruction, the return will be to after the first branch, effectively returning control back to the caller of .setspaceptrfromptr, which is our program. This method of indirect reference allows the first address to be constant across releases of the system, while the second one can be allowed to vary from release to release – the address given is for V4R4M0 – without your program having to be re-compiled. The .setspaceptrfromptr module returns the address of the primary associated space in R3 for the system pointer identified by the address in R4.

To summarize what we have done so far, here is the annotated code:

```

SETSPFP:
    LQ 6, 0X40(30), 11    ; load pointer to R6-R7
    TXER 0, 0, 41        ; if tag bits not set: trap to error

HAVE-TAGGED-POINTER:
    ORI 3, 6, 0           ; copy high-order pointer bits to R3 (i.e. pointer type)
                        ; (R3 is actually not used below, so the copy is redundant)
    ADDI 4, 7, 0          ; copy low-order pointer bits to R4 (i.e. the address)

CHECK-POINTER-TYPE:
    RLDC 7, 6, 2, 62      ; extract two top bits of R6 to R7 and set condition
    BC 4, 2, OXC          ; if pointer were a system pointer
    BLA 0X3801960         ; call SLIC-routine “.setspaceptrfromptr”
                        ; via the system jump table at FF...FF 801960
    ADDI 4, 3, 0          ; copy resulting address of space to R4

HAVE-ADDRESS:            ; R4 is now the address of the associated space

```

<sup>1</sup> in place of ‘NORTHSTAR’, we have also seen ‘Apache’.

The next instruction sets pointer type bits as appropriate for a space pointer:

**E8C08110** LD 6, 0X8110(0)

<b>E</b>	<b>8</b>	<b>C</b>	<b>0</b>	<b>8</b>	<b>1</b>	<b>1</b>	<b>0</b>				
1110	1000	1100	0000	1000	0001	0001	0000				
								; each hex digit expanded into bits			
111010	001110	000000	1000000	100010000					; bits grouped into instruction fields		
LD=58	R6	RO	FF...FF8110							; R6 = doubleword at address FF...FF8110	

## LD - Load Doubleword Instruction

The Load Doubleword instruction

LD Rd, signed displacement (Ra)

loads the doubleword in memory addressed by the effective address into the destination register Rd. The effective address is formed by adding the signed displacement value in the instruction to the contents of register Ra except when the Ra field is zero, in which case the signed displacement is the effective address.

At address FFFFFFFF FF8110 we have this value: 8000000000000000 which is then loaded into R6. These bits are the pointer type bits for a space pointer (which is good as we are trying to build a space pointer).

The next instruction is another powerful, but complicated, rotate instruction:

**788701E5** RLDICR. 7, 4, 0, 39 ; Rotate Left Ra, Rs, SH, MB, then Clear on the Right

<b>7</b>	<b>8</b>	<b>8</b>	<b>7</b>	<b>0</b>	<b>1</b>	<b>E</b>	<b>5</b>		
0111	1000	1000	0111	0000	0001	1110	0101		
								; each hex digit expanded into bits	
011110	00100	00111	00000	00111	001	0	1		
30	R4	R7	*0	*39	1	0	1		
								; bits grouped into instruction fields	
								; R7 = R4 rotl 0, clear 24 bits, record	

## RLDICR Instruction

The “Rotate Left Double Word Immediate then Clear Right (recording)”-instruction:

RLDICR Ra, Rs, SH, ME (Rightmost bit = 0)

RLDICR Ra, Rs, SH, ME (Rightmost bit = 1)

works as follows: the contents of Rs are rotated left the number of bits specified by operand SH. Rotated means that bits shifted out at one end are inserted at the other end. A mask is generated having 1 bits from bit 0 through bit ME and 0 bits elsewhere. The rotated data is ANDed with the generated mask and the result is placed into Ra. Note that both SH and ME are *split fields*, just like SH and MB for the RLDICL instruction, thus SH = 000000 = 0. The ME field is constructed by concatenating bit 26 with bits 21 through 25, thus ME = 100111 = 39.

Note that RLDICR can be used to extract, rotate, shift, and clear bit fields using the methods shown below:

- To extract an  $n$ -bit field, that starts at bit position  $b$  in Rs, right-justified into Ra (clearing the remaining  $64 - n$  bits of Ra), set SH =  $b$  and ME =  $n - 1$ .
- To rotate the contents of a register left by  $n$  bits, set SH =  $n$  and ME = 63; to rotate the contents of a register right by  $n$  bits, set SH =  $(64 - n)$ , and ME = 63.
- To shift the contents of a register left by  $n$  bits, set SH =  $n$  and ME =  $63 - n$ .
- To clear the low-order  $n$  bits of a register, set SH = 0 and ME =  $63 - n$ .

The effect of RLDICR. 7, 4, 0, 39 is to copy R4 to R7, then to clear the low-order  $63 - 39 = 24$  bits of R7. The period after the operation code is the “record result condition” mark causing the CF0 field in the *condition register* to be set according to the resulting value. I’m not sure what we are trying to test here, but my guess is that the following TXER instruction traps if the result was zero, i.e. a *null* pointer. I’ll conduct a small experiment later to find out.

Now we come to a very important instruction:

**7C0103E6**      SETTAG      ; prepare to set tag bits

<b>7</b>	<b>C</b>	<b>0</b>	<b>1</b>	<b>0</b>	<b>3</b>	<b>E</b>	<b>6</b>	
0111	1100	0000	0001	0000	0011	1110	0110	; each hex digit expanded into bits
011111	00000	0000100000	00111110011	0				; bits grouped into instruction fields
31	0	64	499	0				; Set tag bit flag

## SETTAG Instruction

This is another undocumented AS/400 extension. The effect is that the tag bit flag is set in an internal control register causing the tag bits to be set in memory for a Store Quadword instruction (**STQ**) to follow.

The next instruction we can now easily decode. It is an old friend that simply copies register R4 (containing the virtual address of the associated space for the pointer) to R7. Now the register pair R6-R7 contains the bits for the space pointer we want. The final instruction stores the pointer, setting the tag bit in the process:

**F8DE0052**      STQ 6, 0X50(30)      ; store quadword at offset 0X50 from (R30)

<b>F</b>	<b>8</b>	<b>D</b>	<b>E</b>	<b>0</b>	<b>0</b>	<b>5</b>	<b>2</b>	
1111	1000	1101	1110	0000	0000	0101	0010	; each hex digit expanded into bits
111110	00110	11110	00000000010100	10				; bits grouped into instruction fields
STQ=62	R6	R30		50				; quadword at address x50(R30) = R6, R7 pair

## STQ, Store Quadword Instruction

Complementary to the **LQ**-instruction, Store Quadword (**STQ**), stores 16 bytes from two consecutive general-purpose registers into memory at the effective address and turns on the two tag bits, if the tag bit flag in the control register is still set, then resets the tag bit flag (I think). The effective address is formed by adding the signed displacement value in the instruction to the contents of register Ra except when the Ra field is zero, in which case the signed displacement is the effective address. The bit encoding for STQ is the same as for the simple STD (Store Doubleword) except that the low-order two bits are B'10'. These bits can be ignored when the address is calculated, because the quadword must be aligned on a 16-byte boundary.

## Final Annotated SETSPFP Code

After this long and tedious excursion into the boring world of bits we have gained a good understanding of how the SETSPFP instruction is implemented at the machine-code level:

```

SETSPFP:
    LQ 6, 0X40(30), 11      ; load pointer to R6-R7
    TXER 0, 0, 41          ; if tag bits not set: trap to error (pointer not set)

HAVE-TAGGED-POINTER:
    ORI 3, 6, 0             ; copy high-order pointer bits to R3 (i.e. pointer type)
                                ; (R3 is actually not used below, so the copy is redundant)
    ADDI 4, 7, 0            ; copy low-order pointer bits to R4 (i.e. the address)

CHECK-POINTER-TYPE:
    RLDI CL. 7, 6, 2, 62    ; extract two top bits of R6 to R7 and set condition
    BC 4, 2, 0XC           ; if pointer were a system pointer
    BLA 0X3801960          ; call SLIC-routine ".setspaceptrfromptr" via FF...FF 801960
    ADDI 4, 3, 0           ; copy resulting address of space to R4

; R4 is now the address of the associated space

GET-POINTER-TYPE:
    LD 6, 0X8110(0)        ; R6 = space pointer type bits from FF...FF8110

    RLDI CR. 7, 4, 0, 39    ; R7 = R4, then clear low-order 24 bits of R7
    TXER 1, 0, 38          ; if result was zero (?), trap to error routine

POINTER-IS-NOW-VALID:
    SETTAG                 ; set tag bit flag
    ADDI 7, 4, 0           ; R7 = address of space (from R4)

STORE-THE-POINTER:
    STQ 6, 0X50(30)        ; store R6, R7 as space pointer and set tag bit
  
```

## Chapter 7

### Accessing Arbitrary Data in Memory

#### Accessing Arbitrary Data

In this chapter we shall show how easy it is to access just about *any* data in the AS/400 single-level store without any possibility of detection. This is a somewhat scary prospect, as you are not supposed to be able to do that. Many installations rely on assurance that this is not possible at security levels above 30. In order to assess a risk, you should first know what the threat is. This chapter is designed to open your eyes.

#### Set Space Pointer From Any Pointer

We expand the little program from chapter 6 (**MISTPTR1**) to accept a pointer (of *any* type: DCL PTR) as a parameter. The program (**MIMAKPTR**) then proceeds to turn that same pointer into a space pointer to the object pointed to:

```
DCL SPCPTR .PARM1 PARM;  
DCL PTR .PTR BAS(.PARM1); /* any type pointer */  
DCL SPCPTR .SPCPTR BAS(.PARM1); /* note: overlays .PTR */  
DCL OL PARAMETERS (.PARM1) PARM EXT MIN(1);  
  
ENTRY * (PARAMETERS) EXT;  
  SETSPFP .SPCPTR, .PTR; /* set space pointer from pointer */  
  RTX *;
```

We already know how to disassemble the program and to read the RISC code for SETSPFP. The new aspect is that we now need one more reference to access the pointer. The 64-bit address of each parameter is passed to the calling program. Thus, the *address* of .PARM1 pointing to the *pointer* .PARM1 is received.

```
0014C8 000048 E89E0008 LD 4,0X8(30) 01 ; load address of 1st parameter  
0014CC 00004C E3440002 LQ 26,0X0(4),2 ; load pointer .PARM1. Address part to R27  
0014D0 000050 7C0004C8 TXER 0,0,41 ; test if tag bits set
```

The remainder of the code continues as before:

```
0014D4 000054 E0DB000B LQ 6,0X0(27),11 ; load .PTR (R6=type, R7=address)  
0014D8 000058 7C0004C8 TXER 0,0,41 ; test if tag bits set  
  
0014DC 00005C 78DA17A1 RLDICL 26,6,2,62  
0014E0 000060 38870000 ADDI 4,7,0  
0014E4 000064 38660000 ADDI 3,6,0  
0014E8 000068 4082000C BC 4,2,0XC  
0014EC 00006C 4B801963 BLA 0X3801960  
0014F0 000070 38830000 ADDI 4,3,0  
  
0014F4 000074 E8C08110 LD 6,0X8110(0)  
0014F8 000078 788701E5 RLDICR 7,4,0,39  
0014FC 00007C 7C200348 TXER 1,0,38  
001500 000080 7C0103E6 SETTAG  
001504 000084 38E40000 ADDI 7,4,0  
001508 000088 F8DB0002 STQ 6,0X0(27)
```

#### Counterfeiting the Pointer

What we want to do is to *construct* a valid pointer given only the address. Such a pointer is counterfeit because it was not set by the system, but by us. The usual statement is that you cannot counterfeit a pointer. We shall proceed to do just that. If you replace the two instructions at offset 0014D4 and onwards with

```
0014D4 000054 E89B0008 LD 4,0X8(27) ; load address part of .PTR to R4  
0014D8 000058 4280001C BC 20,0,0X1C ; always branch to 0014F4
```

the remainder of the code will complete the counterfeiting process, in particular setting the tag bits for our freshly minted pointer. I'll show you one more time how to interpret the bits in the instruction word:

```
E89B0008 LD 4,0X8(27) ; R4 = doubleword at 8(27)
```

```

E      8      9      B      0      0      0      8
1110 1000 1001 1011 0000 0000 0000 1000 ; each hex digit expanded into bits
111010 00100 11011 0000000000001000    ; bits grouped into instruction fields
LD=58   R4    R27                               8 ; R4 = doubleword at address (R27) + 0x8

```

You use SST to change the code, shown **before** and **after**:

```

                                Display Storage
Control . . . . . nnnnn, Pnnnn, LCCCCC, .CCCCC, >
Address . . . . . 2D2C5B4169 001480

1480 FB41FF33 605D0000 7C0802A6 F8010028 * ....-)...@...8... *
1490 F821FE61 39800001 F9810060 F8410020 * 8../....9..-8... *
14A0 3C005415 F8010008 F82101D8 33FF0060 * ...8...8..Q...- *
14B0 7C2001C8 419A8053 419D8183 3B600082 * @..H.....-... *
14C0 9B6100A8 3BC30000 E89E0008 E3440002 * ./...C..Y...T... *
14D0 7C0004C8 E0DB000B 7C0004C8 78DA17A1 <==== * @..H....@..H.... *
                                |
14D0 7C0004C8 E89B0008 4280001C 78DA17A1 * @..HY..... *
                                |
14E0 38870000 38660000 4082000C 4B801963 * ..... *
14F0 38830000 E8C08110 788701E5 7C200348 * .....Y.....V@... *
1500 7C0103E6 38E40000 F8DB0002 80E10118 * @..W.U..8..... *
1510 78E6FFA3 7C000348 419D81C3 419A8063 * .W..@.....C.... *
1520 382101A0 E8010028 7C0803A6 EB41FF33 * .....Y...@..... *
1530 4E800021 4800000C 60000000 0BE00000 * +.....-..... *
1540 00000000 E3C2E3C2 2D2C5B41 69001550 * ....TBTB..$....& *
1550 01000600 00000000 2D2C5B41 69001480 * .....$. .... *
1560 00000000 00000054 07040104 00000048 * ..... *
1570 009C0003 00000000 000000C0 00000000 * ..... *
F3=Exit      F4=Alter labels      F5=Refresh      F6=Display stack
F9=Pop stack  F10=Push stack      F11=Alter storage F12=Cancel

```

And finally pressing **F11** to make the changes permanent.

## Accessing Encapsulated Program Code

As a first example of how to use our manufactured pointer, the following program (**MITSTPT1**) solves the problem of getting a *space pointer* to a program object. First we declare the argument to **MIMAKPTR**:

```

DCL SPCPTR .ARG1 INIT(POINTER);
DCL DD POINTER CHAR(16) BDRY(16);
  DCL PTR .POINTER DEF(POINTER) POS( 1);

  DCL DD PTR-TYPE CHAR(8) DEF(POINTER) POS( 1);
  DCL DD PTR-SEGMENT CHAR(5) DEF(POINTER) POS( 9);
  DCL DD PTR-OFFSET CHAR(3) DEF(POINTER) POS(14);

DCL OL MIMAKPTR (.ARG1) ARG;
DCL SYSPTR .MIMAKPTR;

DCL DD DATA CHAR(8192) BAS(.POINTER); /* some data to look at */

```

Now, the standard template for resolving a pointer:

```

DCL DD RESOLVE CHAR(34);
  DCL DD RESOLVE-TYPE CHAR( 2) DEF(RESOLVE) POS( 1);
  DCL DD RESOLVE-NAME CHAR(30) DEF(RESOLVE) POS( 3);
  DCL DD RESOLVE-AUTH CHAR( 2) DEF(RESOLVE) POS(33) INIT(X'0000');

```

Then we declare parameters to hold the name and (optionally) the type/subtype of the object to access:

```

DCL SPCPTR .PARAM1 PARAM;
DCL DD PARAM-NAME CHAR(10) BAS(.PARAM1);

DCL SPCPTR .PARAM2 PARAM;
DCL DD PARAM-TYPE CHAR(4) BAS(.PARAM2); /* e.g. '0201' for *PGM */

DCL OL PARAMETERS (.PARAM1, .PARAM2) PARAM EXT MIN(2);
DCL DD NBR-PARMS BIN(2);

ENTRY * (PARAMETERS) EXT;

```

Resolve to our counterfeit pointer program:

RESOLVE-TO-PGM:



```

CPYBLA      RESOLVE-TYPE, X'0201';
CPYBLAP     RESOLVE-NAME, "MIMAKPTR", " ";
RSLVSP      .MIMAKPTR, RESOLVE, *, *;

```

Resolve to the object given as parameter:

```

LOAD-ARGUMENTS-AND-CALL:
STPLLEN     NBR-PARMS; /* get number of parms */
CMPNV(B)    NBR-PARMS, 1/EQ(=+2); /* if only one parm, skip: */
CVTCH       RESOLVE-TYPE, PARM-TYPE; /* convert from HEX digits */
CPYBLAP     RESOLVE-NAME, PARM-NAME, " ";
RSLVSP      .POINTER, RESOLVE, *, *;

CVTHC       MSG-TEXT(1:32), POINTER; /* prepare to show 16-byte pointer as hex */
CALLI       SHOW-MESSAGE, *, .SHOW-MESSAGE;

```

This is what we get:

```

Type reply (if required), press Enter.
From . . . : LSVALGAARD 08/29/00 15:02:58
00000000000000002D2C5B4169000000

```

In Frank Soltis' book "Inside the AS/400, 2<sup>nd</sup> ed." Page 201, we find this description of a resolved system pointer: "A resolved pointer describes the object and the authority the user has to the object; it also provides the address of the object in the single-level store. If we look inside one of these 16-byte pointers, we find it divided into two 8-byte parts. The first part contains the description of the object, and the second part contains the virtual address. The first part of a resolved pointer contains status bits that, among other things, identify whether this is a system, space, data, instruction, or procedure pointer. There is also information in this first part about the object or data accessed by the pointer. For example, a system pointer identifies the MI system object type and the OS/400-defined subtype for the object. [...] The final piece of information contained in the first part of the pointer is the authority to the object. [...] The authority field is used only when the pointer is owned by the operating system in the system state. Pointers created by user programs in the user state do not have the authority field filled in. The information in the first part of the pointer (just described) does not occupy all 8 bytes; it occupies only 4 bytes..."

The above explanation is typical of the half-truths that permeate most published "information" about AS/400 internals. We shall devote a later chapter to setting the record straight. At this point we shall just note that the description of the object (its type only – and not its subtype) is found in the *lower* part of the address instead of in the first part of the pointer. The **02** that we see in the resolved pointer tells us that we are dealing with a program. The top byte of the first part is **00**, denoting a system pointer. The address of the object is formed by setting the low-order 24 bits (the *offset* part) to all zeroes: **2D2C5B4169000000**. This is generally so as all objects are aligned on 16Mb boundaries. We clear the offset with the "Copy Bytes Repeatedly"-instruction (**CPYBREP**), which inserts the character given as the 2<sup>nd</sup> operand into all bytes of the 1<sup>st</sup> operand:

```

CPYBREP     PTR-OFFSET, X'00'; /* clear offset to zeroes */

```

Finally, we call **MIMAKPTR** to fabricate our pointer:

```

CALLX       .MIMAKPTR, MIMAKPTR, *;

CVTHC       MSG-TEXT(1:32), POINTER; /* prepare to show result */
CALLI       SHOW-MESSAGE, *, .SHOW-MESSAGE;

```

Here is the result:

```

Type reply (if required), press Enter.
From . . . : LSVALGAARD 08/29/00 15:02:59
80000000000000002D2C5B4169000000

```

The top byte of the first part is **80**, denoting a space pointer.

## Using the Debugger

The debugger for OPM-programs requires a break point to be set in the program being debugged. The **BRK** statement with a suitable text string as its sole operand accomplishes that:



```

BRK "1"; /* "1" is simple break point identifier */

CPYBLA      MSG-TEXT, DATA; /* try to access the data */
CALLI      SHOW-MESSAGE, *, .SHOW-MESSAGE;

RETURN:
RTX        *;

%INCLUDE SHOWMSG

```

You start the debugger and activate the break point to display the **DATA** variable with the commands, e.g.:

```

====> STRDBG MITSTPT1
====> ADDDBKP STMT(1) PGMVAR((DATA ())) OUTFMT(*HEX)
====> CALL MITSTPT1 PARM(MIMAKPTR '0201')

```

Here is the result of using MIMAKPTR with itself as parameter:

```

                                Display Breakpoint

Statement/Instruction . . . . . : 1 /0009
Program . . . . . : MITSTPTR
Recursion level . . . . . : 1
Start position . . . . . : 1
Format . . . . . : *HEX
Length . . . . . : *DCL

Variable . . . . . : DATA
Type . . . . . : CHARACTER
Length . . . . . : 8000
* . . . . + . . . . 1 . . . . + . . . . * . . . + . . . . 1 . . . . + .
00010010009000012D2C5B4169000000 '°$ N
7001030000000000003C4824E5FF001000 'øÇV
80000201D4C9D4C1D2D7E3D940404040 'øMIMAKPTR
40404040404040404040404040404040 '
4040800000000000000000020FF1C3600 ' ø
812FC25F0AEB00000E6792A4C7000000 'aB-ôÄkuG

```

This is indeed the beginning of the program object as you can verify with SST. Note that MIMAKPTR is in the *user domain* (the 0001 value at offset 6 shows that).

End the debugger with:

```

====> ENDDBG

```

## System Domain Objects

Let's modify our test program to go after a very interesting *system domain* object, namely the Authorized User/Password Table where the encrypted password values are stored. The type/subtype is x"0EC5" and the name is QSYUPTBL. First turn the debugger on:

```

====> STRDBG MITSTPT1
====> ADDDBKP STMT(1) PGMVAR(DATA ()) OUTFMT(*HEX)
====> CALL MITSTPT1 PARM(QSYUPTBL '0EC5')

```

At the break point the debugger shows us:

```

Variable . . . . . : DATA
Type . . . . . : CHARACTER
Length . . . . . : 8000
* . . . . + . . . . 1 . . . . + . . . . * . . . + . . . . 1 . . . . + .
00010210009080003CC3B3AE6B0000000 'øC·p,
C0010000000000000024519880B6000020 '{éqø¶
80000EC5D8E2E8E4D7E3C2D340404040 'øEQSYUPTBL
40404040404040404040404040404040 '
40408000000000FE00000021800404050 ' ø\ &
7E1513B1E0EB00001CDB588953000000 '=f\ôûîë

```

Here we can see that the object in the system domain (the 8000 at offset 6). But when we try to actually access the data (CPYBLA MSG-TEXT, DATA), we get an error (at security level 40 and above):

**Object domain or hardware storage protection violation.**  
Function check. MCH6801 unmonitored by MITSTPT1 at statement 1



Blank





Unfortunately, we cannot add a negative offset to get at the functional space of the object (where the system pointer is pointing to). If you try to, you get a run-time exception that the address is outside of the bounds of the associated space. What we really would like to do is to manufacture a space pointer directly from the system pointer. That problem we just solved in chapter 7 and we'll apply the solution here:

```
CPYBWP      . POINTER, . CURSOR;
CPYBREP     PTR-OFFSET, X'00';
CALLX      . MI MAKPTR, MI MAKPTR, *;
```

And now we can access the modify-timestamp:

```
DCL SPC      MBR-CURSOR BAS(. POINTER);
DCL DD      MBR-CHANGE-TIMESTAMP CHAR(8) DEF(MBR-CURSOR) POS(129);
```

## Converting Dates to Timestamps

Back in chapter 4, we went through the data structures and algorithm needed for converting an MI-timestamp to a more readable date and time. The reverse conversion uses the same data structures and the code is straightforward (but still interesting). A new twist is that we don't want the real century, but only IBM's goofy "century-flag":

```
DCL DD CENTURY      ZND(2,0) DEF(YYYYMMDDHHMMSS) POS(1);
DCL DD CENTURY-FLAG ZND(1,0) DEF(YYYYMMDDHHMMSS) POS(2);
DCL DD CYMMDDHHMMSS CHAR(13) DEF(YYYYMMDDHHMMSS) POS(2);

DCL INSPTR . DATE-TO-TIMESTAMP;
ENTRY      DATE-TO-TIMESTAMP INT;
  ADDN      CENTURY, CENTURY-FLAG, 19;
  SUBN      NBR-YEARS, YEAR, 1925; /* 1st period: 1925, 26, 27, 28 */
  DI VREM   NBR-PERIODS, NBR-YEARS, 4, ADD-YEARS; /* 0 1 2 3 */
  MULT      NBR-DAYS, NBR-PERIODS, 1461;
  MULT      D, ADD-YEARS, 365;
  ADDN(S)   NBR-DAYS, D;
  CPYNV     M, MONTH;
  ADDN(S)   NBR-DAYS, DAYS-ACCUM(M);
  ADDN(S)   NBR-DAYS, DAY;
  CMPNV(B)  ADD-YEARS, 3/NEQ(=+2); /* leap year fiddling */
  CMPNV(B)  MONTH, 2/HI(=+2); /* February is one day */
  SUBN(S)   NBR-DAYS, 1; /* longer in the 3rd year */
  MULT      NBR-SECONDS, NBR-DAYS, 86400;
  MULT      S, HOUR, 60; /* S = minutes */
  ADDN(S)   S, MIN;
  MULT(S)   S, 60; /* S = seconds */
  ADDN(S)   S, SEC;
  ADDN(S)   NBR-SECONDS, S;
  SUBN(S)   NBR-SECONDS, 114955386; /* AUG 23, 1928, 12:03:06pm */
  MULT      TIMESTAMP, NBR-SECONDS, 4096000000;
  DI VREM   BIN-TIME-HI, TIMESTAMP, TWO**32, BIN-TIME-LO;
  B        . DATE-TO-TIMESTAMP;
```

We'll need this conversion because we'll want our API to work in date/time format rather than in 64-bit internal timestamp format.

## Must be System State

Because we are trying to modify a system domain object, we need to (e.g. using SST to) make **MI MBRI NF** a system state program as outlined in chapter 7. The **MI MAKPTR** program that we call should then also be a system state program (or at least an "inherit state"-program). You could avoid this proliferation of system programs by incorporating (the one-line) **MI MAKPTR** program into **MI MBRI NF** and patching the resulting program accordingly. By now, you can do this in stride.

## Parameters

For the final revision of **MI MBRI NF**, we will modify it to accept two parameters; a control block that specifies the operation to perform and the member to perform it on; and an information block with the type, change dates/times, and the descriptive text. We'll also decide to have a separate operation for changing each of the pieces of the information block because this is often what we need in practice. It is rare that we need to change more than one piece at a time. In any case, the code setting up the parameters is trivial boilerplate code that you can easily change to fit your needs.

```

DCL SPCPTR . PARM-CONTROL    PARM;
DCL SPCPTR . PARM-INFO      PARM;

DCL OL PARAMETERS(. PARM-CONTROL, . PARM-INFO) EXT PARM MIN(2);

DCL DD PARM-CONTROL          CHAR(32)  BAS(. PARM-CONTROL);
DCL DD PARM-OPERATION        CHAR(1)    DEF(PARM-CONTROL) POS(1);
/* G - GET INFO FOR MBR */
/* S - SET SOURCE DATE/TIME */
/* O - SET OBJECT DATE/TIME */
/* T - SET MEMBER TYPE */
/* D - SET DESCRIPTIVE TEXT */
DCL DD PARM-FEEDBACK          CHAR(1)    DEF(PARM-CONTROL) POS(2);
/* BLANK - OK */
/* E - ERROR */
/* O - OPERATION UNKNOWN */
DCL DD PARM-LIBRARY           CHAR(10)   DEF(PARM-CONTROL) POS(3);
/* *LIBL - LIBRARY LIST */
DCL DD PARM-FILE              CHAR(10)   DEF(PARM-CONTROL) POS(13);
DCL DD PARM-MEMBER            CHAR(10)   DEF(PARM-CONTROL) POS(23);
/* *FILE - MEMBER = FILE */

DCL DD PARM-INFO              CHAR(86)   BAS(. PARM-INFO);
DCL DD PARM-TYPE              CHAR(10)   DEF(PARM-INFO) POS(1);
DCL DD PARM-DATE-SRC          CHAR(13)   DEF(PARM-INFO) POS(11);
DCL DD PARM-DATE-OBJ          CHAR(13)   DEF(PARM-INFO) POS(24);
DCL DD PARM-TEXT              CHAR(50)   DEF(PARM-INFO) POS(37);

```

## Command **CHGMBR** and Command Processing Program **CHGMBRCL**

This time we'll make a simple command and an associated command processing CL-program to set up the parameters (in particular to allow simple prompting). You can modify these simple programs to suit your needs. This is not rocket science. First the command (**CHGMBR** in file **QCDMSRC**):

```

CMD  PROMPT(' Change File Member Information')
PARM KWD(LIBRARY) TYPE(*CHAR) LEN(10) PROMPT(' Li brary' )
PARM KWD(FILE) TYPE(*CHAR) LEN(10) PROMPT(' File' )
PARM KWD(MEMBER) TYPE(*CHAR) LEN(10) PROMPT(' Member' )
PARM KWD(TYPE) TYPE(*CHAR) LEN(10) PROMPT(' Type' )
PARM KWD(DATESRC) TYPE(*DATE) PROMPT(' Source date' )
PARM KWD(TIMESRC) TYPE(*TIME) PROMPT(' Source time' )
PARM KWD(DATEOBJ) TYPE(*DATE) PROMPT(' Object date' )
PARM KWD(TIMEOBJ) TYPE(*TIME) PROMPT(' Object time' )
PARM KWD(TEXT) TYPE(*CHAR) LEN(50) PROMPT(' Descriptive Text' )
PARM KWD(TEST) TYPE(*CHAR) LEN( 1) PROMPT(' Test, Yes/No' )

```

Create the command:

```
====> CRTCMD CMD(CHGMBR) PGM(CHGMBRCL) REPLACE(*YES)
```

Here is a typical prompt screen:

Change File Member Information (CHGMBR)		
Type choices, press Enter.		
Library . . . . .	> *LIBL	Character value
File . . . . .	> QMLSRC	Character value
Member . . . . .	> MIHELLO	Character value
Type . . . . .	> MI	Character value
Source date . . . . .	> 7/27/2000	Date
Source time . . . . .	> 11:22:33	Time
Object date . . . . .	> 9/02/2000	Date
Object time . . . . .	> 22:33:44	Time
Descriptive Text . . . . .	> 'Hello World'	
Test, Yes/No . . . . .	> Y	Character value

The last parameter is for testing. If “Y” the member information is retrieved and the information block is sent as a message to the requesting job’s message queue. The full command line for the above prompt was:

```
====> CHGMBR LIBRARY(*LIBL) FILE(QMSRC) MEMBER(MIHELLO) TYPE(MI) DATESRC('7/27/
2000') TIMESRC('11:22:33') DATEOBJ('9/02/2000') TIMEOBJ('22:33:44') TEXT('Hello
World') TEST(Y)
```

Only parameters that are given will be changed by the program.

## Command Processing Program

Compile the CL-program **CHGMBRCL** in **QCLSRC** that is to serve as command processing program:

```
PGM PARM(&PARMLIB &PARMFILE &PARMMBR &PARMTYPE &PARMSRC &PARMSRC +
&PARMDOBJ &PARMTOBJ &PARMTEXT &PARMTEST)

DCL &PARMLIB *CHAR LEN(10)
DCL &PARMFILE *CHAR LEN(10)
DCL &PARMMBR *CHAR LEN(10)
DCL &PARMTYPE *CHAR LEN(10)
DCL &PARMSRC *CHAR LEN(07) /* DATE WHEN SOURCE CHANGED */
DCL &PARMSRC *CHAR LEN(06) /* TIME WHEN SOURCE CHANGED */
DCL &PARMDOBJ *CHAR LEN(07) /* DATE WHEN OBJECT CHANGED */
DCL &PARMTOBJ *CHAR LEN(06) /* TIME WHEN OBJECT CHANGED */
DCL &PARMTEXT *CHAR LEN(50)
DCL &PARMTEST *CHAR LEN(1) /* Y/N */

DCL &CONTROL *CHAR LEN(32)
DCL &QUALNAME *CHAR LEN(30)
DCL &INFO *CHAR LEN(86)

CHGVAR &QUALNAME VALUE(&PARMLIB *CAT &PARMFILE *CAT &PARMMBR)
IF (&PARMMBR *EQ ' ') GOTO DONE

IF (&PARMTYPE *EQ '*NONE ') +
THEN(DO)
CHGVAR &CONTROL VALUE('T ' *CAT &QUALNAME)
CHGVAR &INFO VALUE(' ')
CALL PGM(MIMBRINF) PARM(&CONTROL &INFO)
ENDDO
ELSE +
IF (&PARMTYPE *NE ' ') +
THEN(DO)
CHGVAR &CONTROL VALUE('T ' *CAT &QUALNAME)
CHGVAR &INFO VALUE(&PARMTYPE)
CALL PGM(MIMBRINF) PARM(&CONTROL &INFO)
ENDDO

IF (&PARMSRC *NE '0000000') +
THEN(DO)
CHGVAR &CONTROL VALUE('S ' *CAT &QUALNAME)
CHGVAR &INFO VALUE(&PARMTYPE *CAT &PARMSRC *CAT &PARMSRC)
CALL PGM(MIMBRINF) PARM(&CONTROL &INFO)
ENDDO

IF (&PARMDOBJ *NE '0000000') +
THEN(DO)
CHGVAR &CONTROL VALUE('O ' *CAT &QUALNAME)
CHGVAR &INFO VALUE(&PARMTYPE *CAT ' ' +
*CAT &PARMDOBJ *CAT &PARMTOBJ)
CALL PGM(MIMBRINF) PARM(&CONTROL &INFO)
ENDDO

IF (&PARMTEXT *EQ '*NONE ') +
THEN(DO)
CHGVAR &CONTROL VALUE('D ' *CAT &QUALNAME)
CHGVAR &INFO VALUE(&PARMTYPE +
*CAT ' ' +
*CAT ' ')
CALL PGM(MIMBRINF) PARM(&CONTROL &INFO)
ENDDO
ELSE +
IF (&PARMTEXT *NE '*NONE ') +
THEN(DO)
CHGVAR &CONTROL VALUE('D ' *CAT &QUALNAME)
CHGVAR &INFO VALUE(&PARMTYPE +
*CAT ' ' +
*CAT &PARMTEXT)
CALL PGM(MIMBRINF) PARM(&CONTROL &INFO)
ENDDO
```

DONE:



```

IF (&PARMTEST *EQ 'Y') +
  THEN(DO)
    CHGVAR &CONTROL VALUE('G ' *CAT &QUALNAME)
    CALL PGM(MI MBRI NF) PARM(&CONTROL &INFO)
    SNDMSG MSG(&INFO) TOUSR(*REQUESTER)
  ENDDO
ENDPGM

```

When you run the command, with TEST(Y), the test message that is sent looks like this:

```

Type reply (if required), press Enter.
From . . . : LSVALLGAARD 09/06/00 22:31:22
MI 10007271122331000902223344Hello World

```

## The Complete *MI MBRI NF* Program

```

DCL SPCPTR . PARM-CONTROL PARM;
DCL SPCPTR . PARM-INFO PARM;

DCL OL PARAMETERS(. PARM-CONTROL, . PARM-INFO) EXT PARM MIN(2);

DCL DD PARM-CONTROL CHAR(32) BAS(. PARM-CONTROL);
DCL DD PARM-OPERATION CHAR(1) DEF(PARM-CONTROL) POS(1);
/* G - GET INFO FOR MBR */
/* S - SET SOURCE DATE/TIME */
/* O - SET OBJECT DATE/TIME */
/* T - SET MEMBER TYPE */
/* D - SET DESCRIPTIVE TEXT */
DCL DD PARM-FEEDBACK CHAR(1) DEF(PARM-CONTROL) POS(2);
/* BLANK - OK */
/* E - ERROR */
/* O - OPERATION UNKNOWN */
DCL DD PARM-LIBRARY CHAR(10) DEF(PARM-CONTROL) POS(3);
/* *LIBL - LIBRARY LIST */
DCL DD PARM-FILE CHAR(10) DEF(PARM-CONTROL) POS(13);
DCL DD PARM-MEMBER CHAR(10) DEF(PARM-CONTROL) POS(23);
/* *FILE - MEMBER = FILE */

DCL DD PARM-INFO CHAR(86) BAS(. PARM-INFO);
DCL DD PARM-TYPE CHAR(10) DEF(PARM-INFO) POS(1);
DCL DD PARM-DATE-SRC CHAR(13) DEF(PARM-INFO) POS(11);
DCL DD PARM-DATE-OBJ CHAR(13) DEF(PARM-INFO) POS(24);
DCL DD PARM-TEXT CHAR(50) DEF(PARM-INFO) POS(37);

DCL EXCM EXCEPTION-LIST EXCID(H' 0000') BP(ERROR-DETECTED) IGN;

DCL SYSPTR . CONTEXT;
DCL SYSPTR . CURSOR;
DCL SPCPTR . CURSOR-SPACE;
DCL SPC CURSOR-SPACE BAS(. CURSOR-SPACE);
DCL DD CSR-MBR-HEADER BIN(4) DEF(CURSOR-SPACE) POS(5);

DCL SPC MBR-CURSOR BAS(. POINTER);
DCL DD MBR-CHANGE-TIMESTAMP CHAR(8) DEF(MBR-CURSOR) POS(129);

DCL SPCPTR . MBR-HEADER;
DCL SPC MBR-HEADER BAS(. MBR-HEADER);
DCL SYSPTR . MHDR-PREV-MCB DI R;
DCL SYSPTR . MHDR-NEXT-MCB DI R;
DCL SYSPTR . MHDR-FILE-CB DI R;
DCL SYSPTR . MHDR-SHARE-DIR DI R;
DCL SYSPTR . MHDR-DATA-DICT DI R;

DCL DD MHDR-STATUS CHAR(2) DI R;
DCL DD * CHAR(2) DI R;
DCL DD MHDR-TEXT CHAR(50) DI R;
DCL DD MHDR-TYPE CHAR(10) DI R;
DCL DD * CHAR(10) DI R;

DCL DD MHDR-CHANGE-DATE CHAR(13) DI R;
DCL DD MHDR-CREATE-DATE CHAR(13) DI R;

DCL DD MHDR-PREFRD-UNIT CHAR(1) DI R;
DCL DD MHDR-ALLOC-TYPE CHAR(2) DI R;
DCL DD MHDR-INIT-RECS BIN(4) DI R;
DCL DD MHDR-RECS-EXTEND BIN(2) DI R;
DCL DD MHDR-NBR-EXTENDS BIN(2) DI R;
DCL DD MHDR-RECOVER-OPT CHAR(1) DI R;

```

```

DCL DD MHDR-SAVE-DATE CHAR(13) DIR;
DCL DD MHDR-RSTR-DATE CHAR(13) DIR;
DCL DD MHDR-%-DLT-ALLOW CHAR(1) DIR;
DCL DD MHDR.USER-AREA BIN(4) DIR;
DCL DD MHDR-OLD-S-DATE CHAR(13) DIR;
DCL DD MHDR-OLD-R-DATE CHAR(13) DIR;
DCL DD MHDR..... CHAR(1) DIR;

DCL SPCPTR .ARG1 INIT(POINTER);
DCL DD POINTER CHAR(16) BDRY(16);
DCL PTR .POINTER DEF(POINTER) POS( 1);

DCL DD PTR-TYPE CHAR(8) DEF(POINTER) POS( 1);
DCL DD PTR-SEGMENT CHAR(5) DEF(POINTER) POS( 9);
DCL DD PTR-OFFSET CHAR(3) DEF(POINTER) POS(14);

DCL OL MI MAKPTR (.ARG1) ARG;
DCL SYSPTR .MI MAKPTR;

DCL DD RESOLVE CHAR(34);
DCL DD RESOLVE-TYPE CHAR( 2) DEF(RESOLVE) POS( 1) INIT(X' 0000' );
DCL DD RESOLVE-NAME CHAR(30) DEF(RESOLVE) POS( 3);
DCL DD RESOLVE-AUTH CHAR( 2) DEF(RESOLVE) POS(33) INIT(X' 0000' );

/*****/

ENTRY * (PARAMETERS) EXT;
CPYBLA PARM-FEEDBACK, " ";
CMPBLA(B) RESOLVE-TYPE, X' 0000' /NEQ(TEST-LIBRARY);
RESOLVE-TO-MAKE-POINTER-PGM:
CPYBLA RESOLVE-TYPE, X' 0201' ;
CPYBLAP RESOLVE-NAME, "MI MAKPTR", " ";
RSLVSP .MI MAKPTR, RESOLVE, *, *;

TEST-LIBRARY:
CMPBLAP(B) PARM-LIBRARY, "*LIBL", " " /NEQ(GET-CONTEXT);
CPYBWP .CONTEXT, *, /* NULL */
B TEST-MEMBER;

GET-CONTEXT:
CPYBLA RESOLVE-TYPE, X' 0401' ;
CPYBLAP RESOLVE-NAME, PARM-LIBRARY, " ";
RSLVSP .CONTEXT, RESOLVE, *, *;

TEST-MEMBER:
CPYBLAP RESOLVE-NAME( 1: 30), PARM-FILE, " ";
CPYBLA RESOLVE-NAME(11: 10), PARM-FILE;
CMPBLAP(B) PARM-MEMBER, "*FILE", " " /EQ(=+2);
CPYBLA RESOLVE-NAME(11: 10), PARM-MEMBER;

RESOLVE-TO-CURSOR:
CPYBLA RESOLVE-TYPE, X' 0D50' ;
RSLVSP .CURSOR, RESOLVE, .CONTEXT, *;
SETSPFP .CURSOR-SPACE, .CURSOR;
ADDSP .MBR-HEADER, .CURSOR-SPACE, CSR-MBR-HEADER;

TEST-OPERATION:
CMPBLA(B) PARM-OPERATION, "S" /EQ(SET-SOURCE-INFO);
CMPBLA(B) PARM-OPERATION, "T" /EQ(SET-MEMBER-TYPE);
CMPBLA(B) PARM-OPERATION, "D" /EQ(SET-DESCRPTIVE-TEXT);

GET-MBR-OBJECT:
CPYBWP .POINTER, .CURSOR;
CPYBREP PTR-OFFSET, X' 00' ;
CALLX .MI MAKPTR, MI MAKPTR, *;

CMPBLA(B) PARM-OPERATION, "O" /EQ(SET-OBJECT-INFO);
CMPBLA(B) PARM-OPERATION, "G" /EQ(GET-INFO-FOR-MBR);
B OPERATION-ERROR;

GET-INFO-FOR-MBR:
CPYBLA PARM-TYPE, MHDR-TYPE;
CPYBLA PARM-TEXT, MHDR-TEXT;
CPYBLA BIN-TIMESTAMP, MBR-CHANGE-TIMESTAMP;
CALLI TIMESTAMP-TO-DATE, *, .TIMESTAMP-TO-DATE;
CPYBLA PARM-DATE-OBJ, CYMMDDHHMMSS;
CPYBLA PARM-DATE-SRC, MHDR-CHANGE-DATE;
B RETURN;

SET-SOURCE-INFO:

```

```

        CPYBLA      MHDR-CHANGE-DATE, PARM-DATE-SRC;
        B          RETURN;

SET-OBJECT-INFO:
        CPYBLA      CYYMMDDHHMMSS, PARM-DATE-OBJ;
        CALLI      DATE-TO-TIMESTAMP, *, . DATE-TO-TIMESTAMP;
        CPYBLA      MBR-CHANGE-TIMESTAMP, BIN-TIMESTAMP;
        B          RETURN;

SET-MEMBER-TYPE:
        CPYBLA      MHDR-TYPE, PARM-TYPE;
        B          RETURN;

SET-DESCRIPTIVE-TEXT:
        CPYBLA      MHDR-TEXT, PARM-TEXT;
        B          RETURN;

OPERATION-ERROR:
        CPYBLA      PARM-FEEDBACK, "O";
RETURN:
        RTX          *;

ERROR-DETECTED:
        CPYBLA      PARM-FEEDBACK, "E";
        RTX          *;

/***** TIME CONVERSIONS *****/

DCL DD BIN-TIMESTAMP CHAR(8) BDRY(8);
DCL DD BIN-TIME-HI BIN(4) UNSGND DEF(BIN-TIMESTAMP) POS(1);
DCL DD BIN-TIME-LO BIN(4) UNSGND DEF(BIN-TIMESTAMP) POS(5);

DCL DD TIMESTAMP      PKD(21,0); /* CAN HOLD 64-BIT UNSIGNED */
DCL DD TIMESTAMP-HI   PKD(11,0);
DCL DD TIMESTAMP-LO   PKD(11,0);
DCL DD TWO**32        PKD(11,0) INIT(P' 4294967296');
DCL DD NBR-SECONDS    PKD(15,0);
DCL DD NBR-DAYS       BIN(4);
DCL DD NBR-YEARS      BIN(4);
DCL DD ADD-YEARS      BIN(4);
DCL DD NBR-PERIODS    BIN(4);
DCL DD DAY-MONTH      BIN(4);
DCL DD D              BIN(4);
DCL DD S              BIN(4);
DCL DD M              BIN(4);
/* DAY BASE FOR: JanFebMarAprMayJunJul AugSepOctNovDec*/
DCL DD DAYS CHAR(36) INIT("000031059090120151181212243273304334");
DCL DD DAYS-ACCUM (12)ZND(3,0) DEF(DAYS) POS(1);

DCL DD YYYYMMDDHHMMSS CHAR(14);
DCL DD YEAR          ZND(4,0) DEF(YYYYMMDDHHMMSS) POS( 1);
DCL DD MONTH         ZND(2,0) DEF(YYYYMMDDHHMMSS) POS( 5);
DCL DD DAY           ZND(2,0) DEF(YYYYMMDDHHMMSS) POS( 7);
DCL DD HOUR          ZND(2,0) DEF(YYYYMMDDHHMMSS) POS( 9);
DCL DD MIN           ZND(2,0) DEF(YYYYMMDDHHMMSS) POS(11);
DCL DD SEC           ZND(2,0) DEF(YYYYMMDDHHMMSS) POS(13);

DCL DD CENTURY       ZND(2,0) DEF(YYYYMMDDHHMMSS) POS(1);
DCL DD CENTURY-FLAG  ZND(1,0) DEF(YYYYMMDDHHMMSS) POS(2);
DCL DD CYYMMDDHHMMSS CHAR(13) DEF(YYYYMMDDHHMMSS) POS(2);

DCL INSPTR . TIMESTAMP-TO-DATE;
ENTRY      TIMESTAMP-TO-DATE INT;
        CPYNV      TIMESTAMP-LO, BIN-TIME-LO;
        CPYNV      TIMESTAMP-HI, BIN-TIME-HI;
        MULT       TIMESTAMP, TIMESTAMP-HI, TWO**32;
        ADDN(S)    TIMESTAMP, TIMESTAMP-LO;
        DIV(SR)    TIMESTAMP, 4096000000; /* NOW SECONDS */
        ADDN(S)    TIMESTAMP, 43386; /* 12:03:06 PM */
        DIVREM     NBR-DAYS, TIMESTAMP, 86400, NBR-SECONDS;
        SUBN(S)    NBR-DAYS, 131; /* WAS: AUG 23, 1928, NOW: JAN 01, 1929 */
        DIVREM     NBR-PERIODS, NBR-DAYS, 1461, NBR-DAYS; /* 4 YEARS */
        MULT       NBR-YEARS, NBR-PERIODS, 4;
        ADDN(S)    NBR-YEARS, 1929;
        DIVREM     ADD-YEARS, NBR-DAYS, 365, NBR-DAYS;
        ADDN       YEAR, NBR-YEARS, ADD-YEARS;
        CPYNV      M, 13;
        CMPNV(B)   ADD-YEARS, 3/LO(FIND-MONTH);
LEAP-YEAR:
        CMPNV(B)   NBR-DAYS, 59/LO(FIND-MONTH), EQ(FEB-29TH);
        SUBN(S)    NBR-DAYS, 1;

```

```

FIND-MONTH:
  SUBN(S)      M, 1;
  SUBN(B)      DAY-MONTH, NBR-DAYS, DAYS-ACCUM(M)/NEG(FIND-MONTH);
  ADDN         DAY, DAY-MONTH, 1;
  CPYNV(B)     MONTH, M/NNAN(COMPUTE-TIME);
FEB-29TH:
  CPYNV        MONTH, 2;
  CPYNV        DAY, 29;

COMPUTE-TIME:
  DIVREM       HOUR, NBR-SECONDS, 3600, NBR-SECONDS;
  DIVREM       MIN, NBR-SECONDS, 60, SEC;
  SUBN         CENTURY-FLAG, CENTURY, 19;
  B            .TIMESTAMP-TO-DATE;

DCL INSPTR .DATE-TO-TIMESTAMP;
ENTRY      DATE-TO-TIMESTAMP INT;
  ADDN      CENTURY, CENTURY-FLAG, 19;
  SUBN      NBR-YEARS, YEAR, 1925;
  DIVREM    NBR-PERIODS, NBR-YEARS, 4, ADD-YEARS;
  MULT      NBR-DAYS, NBR-PERIODS, 1461;
  MULT      D, ADD-YEARS, 365;
  ADDN(S)   NBR-DAYS, D;
  CPYNV     M, MONTH;
  ADDN(S)   NBR-DAYS, DAYS-ACCUM(M);
  ADDN(S)   NBR-DAYS, DAY;
  CMPNV(B)  ADD-YEARS, 3/NEQ(=+2);
  CMPNV(B)  MONTH, 2/HI(=+2);
  SUBN(S)   NBR-DAYS, 1;
  MULT      NBR-SECONDS, NBR-DAYS, 86400;
  MULT      S, HOUR, 60;
  ADDN(S)   S, MIN;
  MULT(S)   S, 60;
  ADDN(S)   S, SEC;
  ADDN(S)   NBR-SECONDS, S;
  SUBN(S)   NBR-SECONDS, 114955386; /* AUG 23, 1928, 12:03:06 */
  MULT      TIMESTAMP, NBR-SECONDS, 4096000000;
  DIVREM    BIN-TIME-HI, TIMESTAMP, TWO**32, BIN-TIME-LO;
  B         .DATE-TO-TIMESTAMP;

```

Blank

## The Work Control Block Table

In the 1960's, Per Brinch Hansen and I wrote the multiprogramming operation system for the RC/4000 computer (see *e.g.* <http://www.deas.harvard.edu/courses/cs261/papers/hansen70.pdf>). This was a classic early microkernel, based on message-passing between processes, where most of the work involved with I/O was not part of the kernel, but rather, was handled by processes to which I/O request messages were passed. Many later computer operating systems, including the operating systems for S/38 and the AS/400, were, in effect, confused re-implementations of this system. On the AS/400 at the lowest level (even below the MI) we have message-passing *tasks* as the fundamental dispatchable units of work. On top of them, *processes* at the MI-level become focal points for work and resource management. And finally, on top of them again, OS/400 introduces the notion of a *job*. Thus from a user's perspective, jobs are the important entities. On the AS/400, jobs exist as long as they are either running or have spooled output that has not been released. A list of all these jobs is kept in Work Control Block Tables. In this chapter we'll look at how to scan these tables to find information about jobs. For definiteness we'll solve the following problem: "what is the active job (if any) using a given device".

An AS/400 fully qualified external job name is 26-bytes long and consists of a 10-character ‘device or job name’, a 10-character user profile name, and a six-digit sequence number:

The device name is typically something like “**QPADEVxxxx**” where “**xxxx**” is an identifier from 0001 and up. Type the **DSPJOB** command to see the device name and the job name as well for the current session.

Each job has a *Work Control Block* (WCB) entry. There are also ‘defunct’ entries and unused entries in the WCB tables. The tables can be quite large with many thousands of entries. A running job has an associated “Process Communication Object”, the PCO. In the PCO, the second pointer is a system pointer to the WCB table containing the job:

On a large system there can be several WCB tables. There is a master (or *root*) table in the **QWCBT00** object in **QSYS**, that contains pointers to the other WCB tables. The PCO of a job on the AS/400 contains a pointer to **QWCBT00** at absolute offset x'1D0' (or x'1B0' inside the space):

There is a special MI-keyword (**BASPCO**) that allows a data structure to be *based* on the PCO, and we can define a SYSPTR data-item with in this structure to represent the QWCBT00 pointer:

9-1

You can then get a space pointer to the data space portion using the standard **SETSPFP**-instruction:

```
GET-WCB-ROOT:
    SETSPFP    .WCB-ROOT, @WCBT00;
```

If you disassemble a program in SST with the above instruction you'll see that it translates into this RISC code:

```
4B801633      BLA 0x3801630      ; get address of PCO to R3
E30301BB      LQ 24,0x1B0(3),11 ; load pointer at offset 1B0 (i.e. @WCBT00)
7C0004C8      TXER 0,0,41      ; continue with standard check
...           ; ...
FB1E0052      STQ 24,0x50(30)   ; store resulting space pointer
```

In chapter 6 we learned about the Branch with Link to Absolute address (**BLA**) instruction. At the address given in the instruction we find:

```
Address FFFFFFFF 801630:
801630 782301E4 RLDICR 3,1,0,39 ; R3 = R1 with lower 24 bits cleared
801634 38630380 ADDI 3,3,896 ; R3 = R3 + x'380'
801638 E8630000 LD 3,0x0(3) ; follow chain to
80163C E8630020 LD 3,0x20(3) ; get address of PCO
801640 4E800020 BCLR 20,0 ; unconditional return to link register
```

The details of this are a subject of a later chapter. The all-important register here is **R1** that always holds the address of the current invocation stack frame. The main point is that the **BASPCO** clause causes a call to the **SLIC** routine to get the PCO address. There is some confusion between the terms Process Communication Object (PCO) and Process Control Space (PCS). The later is the space containing the object. There is a difference in address of 32 (x '20').

## The Master (Root) WCB Table List

Here is the beginning of QSYS/QWCBT00 object (type/subtype: 19D0):

```
Address 0E40F74DB5 000000
000000 00010008 00808000 0E40F74D B5000000 .....00.. 7($...
000010 C0010000 00000000 0E40F74D B5000100 {..... 7($... ← address of space
000020 800019D0 D8E6C3C2 E3F0F040 40404040 0..}QWCBT00
000030 40404040 40404040 40404040 40404040
000040 40408000 00000F00 00000008 3F100301 0.....
...
```

Note the address of the associated space at offset x'18'. This introduces us to the notion of an *absolute* offset as the offset relative to the beginning (offset x'000000') of the segment. At absolute offset x'340' (or x'240' within the associated space) is the *master list* of WCB tables:

```
000300 00000000 00000000 00000000 00000000 .....
000310 00008000 00000000 D54E7355 68001A3F ..0....N·EiC... Proc Ctrl
000320 00008000 00000000 2E8B5BEE B8000E3F ..0....»$0½... Index
000330 00008000 00000000 CFBC6BD7 E9000AFF ..0....0",PZ... Queue
000340 00008000 00000000 1D5CC0F2 4B00193F ..0....*{2... Space 1st QWCBTB01 ←
000350 00000000 00000000 00000000 00000000 ..... 2nd QWCBTB02
000360 00000000 00000000 00000000 00000000 ..... 3rd QWCBTB03
000370 00000000 00000000 00000000 00000000 ..... 4th QWCBTB04
000380 00000000 00000000 00000000 00000000 ..... 5th QWCBTB05
000390 00000000 00000000 00000000 00000000 ..... 6th QWCBTB06
0003A0 00000000 00000000 00000000 00000000 ..... 7th QWCBTB07
0003B0 00000000 00000000 00000000 00000000 ..... 8th QWCBTB08
0003C0 00000000 00000000 00000000 00000000 ..... 9th QWCBTB09
0003D0 00000000 00000000 00000000 00000000 ..... 10th QWCBTB10
0003E0 00000000 00000000 00000000 00000000 ..... 11th QWCBTB11
0003F0 00000000 00000000 00000000 00000000 ..... 12th QWCBTB12
000400 00000000 00000000 00000000 00000000 ..... 13th QWCBTB13
000410 00000000 00000000 00000000 00000000 ..... 14th QWCBTB14
000420 00000000 00000000 00000000 00000000 ..... 15th QWCBTB15
...
0004E0 00000000 00000000 00000000 00000000 ..... 27th QWCBTB27
0004F0 00000000 00000000 00000000 00000000 ..... 28th QWCBTB28
000500 00000000 00000000 00000000 00000000 ..... 29th QWCBTB29
000510 00000000 00000000 00000000 00000000 ..... 30th QWCBTB30
000520 00008000 00000000 FFF36113 05001A3F ..0....3/. Proc Ctrl
000530 00008000 00000000 F233D22E 48001A3F ..0....2·K·ç... Proc Ctrl
000540 00008000 00000000 E4744349 8B001A3F ..0....UEäñ»... Proc Ctrl
```

```
000550 00008000 00000000 EF26CF92 83001A3F ..0.....õ·õkc... Proc Ctrl
```

In the current form of the QWCBT00 object there is only room for pointers to 30 tables. Only 10 are currently used in OS/400 releases so far. Here is the data structure needed to access the tables. Note how in MI we define an *array* of 30 entries:

```
DCL DD WCB-ROOT CHAR(2048) BAS(.WCB-ROOT);
DCL SYSPTR .WCB-TABLES(30) DEF(WCB-ROOT) POS(577); /* relative offset 240 */
```

## Scanning the WCB Tables

The strategy is now simple: for each table, scan for an active job associated with the device. Stop when the first such match is found; the pointer to the table is null; or all 30 potential tables have been searched. To process each pointer to a table, we'll create and call an internal subroutine to search the table:

```
GET-WCB-ROOT:
  SETSPFPF .WCB-ROOT, @WCBT00;

  CPYV THE-TABLE, 0;
SEARCH-NEXT-TABLE:
  ADDN(S) THE-TABLE, 1;
  CMPPTRT(B) .WCB-TABLES(THE-TABLE), * /EQ(RETURN);
  SETSPFPF .WCBSPC, .WCB-TABLES(THE-TABLE); /* get associated space */
  CALLI SEARCH-WCB-TABLE, *, .SEARCH-WCB-TABLE; /* returns if found */
  CMPNV(B) THE-TABLE, 30/NHI(SEARCH-NEXT-TABLE);

RETURN:
  RTX *;
```

## Testing if a Pointer is Null

We used the “Compare Pointer Type”-instruction (**CMPPTRT**) to compare the pointer to a null operand (“\*”). If the result is an Equal condition, the pointer was null. This was taken to signal the end of the list, assuming that there is no “hole” in the list. Note the space between “\*” and “/” to forestall problems with the end-of-comment symbol “\*/”.

## Searching a Work Control Block Table

To search the WCB tables, we will need to know how big each table is. The size of a given Work Control Block Table is at character position 21 within the associated space of the table object:

```
Address 1D5CC0F24B 000000
000000 00011388 00808000 1D5CC0F2 4B000000 ...h·00...*{2...
000010 C0010000 00000000 1D5CC0F2 4B000100 {.....*{2...
000020 800019D0 D8E6C3C2 E3F0F140 40404040 0...}QWCBT01
...
000100 00008000 00000000 0E40F74D B500193F ..0..... 7($... Space
000110 00269700 00270300 00010000 00000000 ..p..... size ←
000120 00000000 00000000 00000000 00000000 .....
000130 00000000 00000000 00000000 00000000 .....
```

```
DCL DD WCBTBL-SPACE CHAR(256) BAS(.WCBSPC);
DCL DD WCBTBL-SIZE BIN(4) DEF(WCBTBL-SPACE) POS(21);
```

Entries in the WCBT are each x'400' bytes long and the first entry starts at x'300' bytes into the space. Here is the beginning of the first one (and an interesting one at that):

Address	1D5CC0F24B 000400	Search for	Hex
000400	E2C3D7C6 40404040	4040D8E2 E8E24040	SCPF QSYS
000410	40404040 F0F0F0F0	F0F00000 000000E8	000000.....Y
000420	00008000 00000000	D1072E57 E6001A3F	..0.....J·iw... Proc Ctrl
000430	00008000 00000000	14066F69 2C0018FF	..0.....?N... Queue Spc
000440	00000000 00000000	16428B3A 7800195F	.....â»·î· Space

Way down, we may find the WCB entry for my interactive job:

```
Address 1D5CC0F24B 01A000
01A000 D3C5C9C6 40404040 4040D3E2 E5C1D3C7 LEIF LSVALG device user
01A010 C1C1D9C4 F1F3F8F6 F0F00000 000000E8 AARD138600.....Y nbr
```



```

01A020 00008000 00000000 E1D6503F 94001A3F ..0.....O&.m... Proc Ctrl
01A030 00008000 00000000 209B60E9 A60018FF ..0.....o-Zw... Queue Spc
01A040 00000000 00000000 0263CDA0 7A00195F .....Αμ:.. Space
01A050 00008000 00000000 08CABD34 CB001938 ..0.....-·ô... Space
01A060 C9200000 00000000 00000002 61000000 I...../... type,status,group
01A070 00008000 00000000 2FB72A8E 3C00193F ..0.....¼·p... Space

```

Here is an MI-structure explaining the fields in a WCB entry that we are interested in:

```

DCL DD WCB-ENTRY CHAR(1024) BAS(.WCB-ENTRY);
DCL DD WCB-DEVICE CHAR(10) DEF(WCB-ENTRY) POS( 1);
DCL DD WCB-USER CHAR(10) DEF(WCB-ENTRY) POS( 11);
DCL DD WCB-NUMBER CHAR( 6) DEF(WCB-ENTRY) POS( 21);
DCL SYSPTR .WCB-PCS DEF(WCB-ENTRY) POS( 33);
DCL DD WCB-TYPE CHAR( 1) DEF(WCB-ENTRY) POS( 97);
DCL DD WCB-STATUS CHAR( 2) DEF(WCB-ENTRY) POS( 98);
DCL DD WCB-GROUP CHAR( 1) DEF(WCB-ENTRY) POS(100);

```

## Scanning WCB for Device Name

We shall take the approach of looping through the table looking for jobs that match our device name. We write this as an internal subroutine:

```

DCL INSPTR .SEARCH-WCB-TABLE;
ENTRY SEARCH-WCB-TABLE INT;

PREPARE-TO-SEARCH-WCB-TABLE:
SETSPFPF .WCB-ENTRY, .WCBSPC;
CPYV(B) THE-OFFSET, H'0300' / POS(=+2); /* first entry */

NEXT-WCBTBL-ENTRY:
ADDN(S) THE-OFFSET, H'0400'; /* size of entry */
CMPNV(B) THE-OFFSET, WCBTBL-SIZE/NLO(.SEARCH-WCB-TABLE);
SETSPPO .WCB-ENTRY, THE-OFFSET;

```

To iterate through the list of entries we have the choice of either adding the size of each entry to the space pointer, or to update our own offset variable (THE-OFFSET) and *inserting* it in the pointer with the “Set Space Pointer Offset”-instruction (**SETSPPO**). You can see from the code above which one we decided to use.

If the device name in the WCB-entry does not match the name of the virtual terminal we quickly skip to the next entry:

```

CHECK-WCBTBL-ENTRY:
CMPBLA(B) PARM-DEVICE, WCB-DEVICE/NEQ(NEXT-WCBTBL-ENTRY);

```

HAVE-DEVICE-MATCH:

For every matching entry, we must check the job status. We are only interested in *active* interactive jobs, and even so must check further status bits to find the job that is actually using the virtual terminal at the moment. It must be said that this information is itself somewhat volatile, as the status of the job can change at any time after we have determined it. We shall assume that this complication can be ignored.

## Checking the Job Status

The WCB entry contains these fields of interest:

```

DCL DD WCB-TYPE CHAR( 1) DEF(WCB-ENTRY) POS( 97);
DCL DD WCB-STATUS CHAR( 2) DEF(WCB-ENTRY) POS( 98);
DCL DD WCB-GROUP CHAR( 1) DEF(WCB-ENTRY) POS(100);

```

The WCB-**TYPE** is “I” for an interactive job.

The WCB-**STATUS** is x'0200' for an active job:

```

Bit 0 = Job being read by spool reader
Bit 1 = Job on job queue
Bit 2 = Job active
Bit 3 = Job completed
Bits 4-15 = Reserved (binary 0)

```

The WCB-**GROUP** has these bits of interest to us:

Bit 5 = **Group job**  
 Bit 6 = **Suspended job**

So, we check the status, group-bits, and type, and go to the next WCB entry, if the job is not active, is a suspended group-job, or is not an interactive job:

```
HAVE-DEVICE-MATCH:
CMPBLA(B)      WCB-STATUS, X'20' /NEQ (NEXT-WCBTBL-ENTRY); /*ACTIVE */
TSTBUM(B)      WCB-GROUP , X'06' /ONES (NEXT-WCBTBL-ENTRY); /*SUSPGRP */
CMPBLA(B)      WCB-TYPE , "I" /NEQ (NEXT-WCBTBL-ENTRY); /*INTERACT*/
```

Note the wonderfully named “Test Bits Under Mask”-instruction (**TSTBUM**). It basically ANDs the leftmost byte of the 1<sup>st</sup> operand with the leftmost (or in this case, single) byte given as the 2<sup>nd</sup> operand. You can test if the result is all ONES, all zeroes (ZER), or mixed (MXD).

To check if the job has been suspended due to the *system request* key being pressed (and therefore not any longer is the truly active job) we have to check a bit in the Process Control Space for the job. Recall that we can get a system pointer to the PCS from the WCB entry:

```
DCL DD WCB-ENTRY CHAR(1024) BAS(.WCB-ENTRY);
DCL DD WCB-DEVICE CHAR(10) DEF(WCB-ENTRY) POS( 1);
DCL DD WCB-USER CHAR(10) DEF(WCB-ENTRY) POS( 11);
DCL DD WCB-NUMBER CHAR( 6) DEF(WCB-ENTRY) POS( 21);
DCL SYSPTR .WCB-PCS DEF(WCB-ENTRY) POS( 33);
```

For completeness, we mention that the type, status, and group bits can also be found in the PCS (at absolute offset x'240'):

000220	00000000	00000000	20474827	1400195F	.....â..... Space
000230	00008000	00000000	08CABD34	CB001938	..ø.....ô... Space
000240	<b>C9200000</b>	84000000	00000002	61000000	I...d...../... <b>type, status, group</b>
000250	00008000	00000000	260CC342	8B00193F	..ø.....Câ... Space

If bit 0 of the byte at absolute offset x'3F3' is set, the job is suspended by a system request:

0003D0	00000078	001AB300	00020000	E2800400	...ï.....Sø..
0003E0	00000000	00000000	00000000	00000000	.....
0003F0	0000C1 <b>00</b>	40404040	00000000	00000000	..A..... <b>system request bit</b>
000400	00000000	00000000	00000000	00000000	.....

The code to check this is straightforward:

```
DCL SPCPTR .WCB-PCO;
DCL DD WCB-PCO CHAR(992) BAS(.WCB-PCO);
DCL DD WCB-PCO-GROUP-STATUS CHAR(1) DEF(WCB-PCO) POS(548);
DCL DD WCB-PCO-SYSREQ-STATUS CHAR(1) DEF(WCB-PCO) POS(980);

SETSPFPF .WCB-PCO, .WCB-PCS;
TSTBUM(B) WCB-PCO-SYSREQ-STATUS, X'80' /NZER(NEXT-WCBTBL-ENTRY);
```

If the bit is off, we have the truly active job (at least at this very moment):

```
HAVE-THE-ACTIVE-ENTRY:
CPYBLA PARM-JOB-DEVICE, WCB-DEVICE;
CPYBLA PARM-JOB-USER, WCB-USER;
CPYBLA PARM-JOB-NBR, WCB-NUMBER;
B RETURN;
```

## Putting It All Together

Assembling the pieces of code into the complete program (**MIWCBSCN**) yields the code below.

```
/* CALL PGM(MIWCBSCN) PARM(DEVICE JOBNAME) */
/* INPUT:  DEVICE CHAR(10) DEVICE NAME FOR TERMINAL */
/* OUTPUT: JOBNAME CHAR(26) ACTIVE JOB USING THE DEVICE */
/* IF NO JOB IS USING THE DEVICE, RETURNS SPACES */

DCL DD SPCPTRS CHAR(48) BDRY(16);
DCL SPCPTR .WCBSPC DEF(SPCPTRS) POS( 1);
DCL SPCPTR .WCB-ROOT DEF(SPCPTRS) POS(17);
DCL SPCPTR .WCB-ENTRY DEF(SPCPTRS) POS(33);

DCL DD OWN-PCO CHAR(512) BASPCO;
DCL SYSPTR @WCBT00 DEF(OWN-PCO) POS(433);
```

```

DCL DD WCB-ROOT CHAR(2048) BAS(.WCB-ROOT);
DCL SYSPTR .WCB-TABLES(30) DEF(WCB-ROOT) POS(577);

DCL DD THE-TABLE BIN(4);
DCL DD THE-OFFSET BIN(4);

DCL DD WCBTBL-SPACE CHAR(256) BAS(.WCBSPC);
DCL DD WCBTBL-SIZE BIN(4) DEF(WCBTBL-SPACE) POS(21);

DCL DD WCB-ENTRY CHAR(1024) BAS(.WCB-ENTRY);
DCL DD WCB-DEVICE CHAR(10) DEF(WCB-ENTRY) POS( 1);
DCL DD WCB-USER CHAR(10) DEF(WCB-ENTRY) POS( 11);
DCL DD WCB-NUMBER CHAR( 6) DEF(WCB-ENTRY) POS( 21);
DCL SYSPTR .WCB-PCS DEF(WCB-ENTRY) POS( 33);
DCL DD WCB-TYPE CHAR( 1) DEF(WCB-ENTRY) POS( 97);
DCL DD WCB-STATUS CHAR( 2) DEF(WCB-ENTRY) POS( 98);
DCL DD WCB-GROUP CHAR( 1) DEF(WCB-ENTRY) POS(100);

DCL SPCPTR .WCB-PCO;
DCL DD WCB-PCO CHAR(992) BAS(.WCB-PCO);
DCL DD WCB-PCO-GROUP-STATUS CHAR(1) DEF(WCB-PCO) POS(548);
DCL DD WCB-PCO-SYSREQ-STATUS CHAR(1) DEF(WCB-PCO) POS(980);

DCL SPCPTR .PARM1 PARM;
DCL DD PARM-DEVICE CHAR(10) BAS(.PARM1);

DCL SPCPTR .PARM2 PARM;
DCL DD PARM-JOB CHAR(26) BAS(.PARM2);
DCL DD PARM-JOB-DEVICE CHAR(10) DEF(PARM-JOB) POS( 1);
DCL DD PARM-JOB-USER CHAR(10) DEF(PARM-JOB) POS(11);
DCL DD PARM-JOB-NBR CHAR( 6) DEF(PARM-JOB) POS(21);

DCL OL PARAMETERS(.PARM1, .PARM2) EXT PARM MIN(2);

/*****/

ENTRY * (PARAMETERS) EXT;
CPYBREP PARM-JOB, " ";

GET-WCB-ROOT:
SETSPFPF .WCB-ROOT, @WCBT00;

CPYNV THE-TABLE, 0;
SEARCH-NEXT-TABLE:
ADDN(S) THE-TABLE, 1;
CMPPTRT(B) .WCB-TABLES(THE-TABLE), */EQ(RETURN);
SETSPFPF .WCBSPC, .WCB-TABLES(THE-TABLE);
CALLI SEARCH-WCB-TABLE, *, .SEARCH-WCB-TABLE;
CMPNV(B) THE-TABLE, 30/NHI(SEARCH-NEXT-TABLE);

RETURN:
CPYBLAP MSG-TEXT, PARM-JOB, " "; /* this is for test only */
CALLI SHOW-MESSAGE, *, .SHOW-MESSAGE; /* this is for test only */
RTX *;

DCL INSPTR .SEARCH-WCB-TABLE;
ENTRY SEARCH-WCB-TABLE INT;

PREPARE-TO-SEARCH-WCB-TABLE:
SETSPFPF .WCB-ENTRY, .WCBSPC;
CPYNV(B) THE-OFFSET, H'0300'/POS(=+2);

NEXT-WCBTBL-ENTRY:
ADDN(S) THE-OFFSET, H'0400';:
CMPNV(B) THE-OFFSET, WCBTBL-SIZE/NLO(.SEARCH-WCB-TABLE);
SETSPPO .WCB-ENTRY, THE-OFFSET;

CHECK-WCBTBL-ENTRY:
CMPBLA(B) PARM-DEVICE, WCB-DEVICE/NEQ(NEXT-WCBTBL-ENTRY);

HAVE-DEVICE-MATCH:
CMPBLA(B) WCB-STATUS, X'20'/NEQ(NEXT-WCBTBL-ENTRY);/*ACTIVE */
TSTBUM(B) WCB-GROUP, X'06'/ONES(NEXT-WCBTBL-ENTRY);/*SUSGRP */
CMPBLA(B) WCB-TYPE, "I"/NEQ(NEXT-WCBTBL-ENTRY);/*INTERACT*/

SETSPFPF .WCB-PCO, .WCB-PCS;
TSTBUM(B) WCB-PCO-SYSREQ-STATUS, X'80'/NZER(NEXT-WCBTBL-ENTRY);

HAVE-ACTIVE-ENTRY:
CPYBLA PARM-JOB-DEVICE, WCB-DEVICE;

```

```

CPYBLA      PARM-JOB-USER,  WCB-USER;
CPYBLA      PARM-JOB-NBR,   WCB-NUMBER;
B           RETURN;

%INCLUDE SHOWMSG      /* test only */

```

Because the program uses the SETSPFPF MI-instruction to get the space pointer to a system domain object (i.e. the WCB tables) it must be a system-state program to run at security levels above 30.

## Performance

Scanning the WCB table is where the overwhelming time is spent. My tests on a 170-based system with a WCB table size of 2435 entries showed that it took 0.16 seconds to search the whole table for a non-existing device. On the average it should take half that, or 0.08 seconds, to find an active job. Most of the time it will take considerably less than that, as the system tries to keep active jobs near the beginning of the table.

These times are fairly long and clearly the system must have a better way of finding jobs. Finding a faster method, perhaps even the method the system uses, will be the subject of the following section. It stands to reason that one would like to be able to search for jobs based on various criteria, the device name, the user profile name, etc. The standard way of doing something like this is to build an index with a key consisting of various permutations of the search criteria.

## A Faster Way of Locating Jobs

Within the **QWCBT00** at absolute offset x'320' (or x'220' inside the associated space) is a system pointer to a *machine index* object (again a major subject for a later chapter):

```

Address 0E40F74DB5 000000
000000 00010008 00808000 0E40F74D B5000000  ....00.. 7($...
000010 C0010000 00000000 0E40F74D B5000100  {...}... 7($...
000020 800019D0 D8E6C3C2 E3F0F040 40404040 0..}QWCBT00
000030 40404040 40404040 40404040 40404040
...
000310 00008000 00000000 D54E7355 68001A3F  ..0.....N·EíÇ... Proc Ctrl
000320 00008000 00000000 2E8B5BEE B8000E3F  ..0.....»$0½... Index ←
000330 00008000 00000000 CFBC6BD7 E9000AFF  ..0.....õ-,PZ... Queue
000340 00008000 00000000 1D5CC0F2 4B00193F  ..0.....*{2... Space

```

The index is **QWCBT\_JOB\_INDEX** (type/subtype x'0EA4') in QSYS:

```

Address 2E8B5BEEB8 000000
000000 00010710 00908000 2E8B5BEE B8000000  ....°0..»$0½...
000010 C0010000 00000000 00000000 FF000000  {...}.....
000020 80000EA4 D8E6C3C2 E36DD1D6 C26DC9D5 0..uQWCBT_JOB_IN
000030 C4C5E740 40404040 40404040 40404040  DEX
000040 40404000 00000000 00000710 00404050  ..... &
000050 80A74E22 E3498000 1CDB5889 53000000 0x·Tñ0..ûië...
000060 00000000 00000000 0F7056C0 A9000000  .....0i{z...
000070 2E8B5BEE B8000000 00020000 00000000  ..»$0½.....
000080 80CF2124 31260000 00000000 00000000 0õ..... ...

```

## Format of the Job Index Header

The following dump, using DMPSYSOBJ, shows parts of the Job Index:.

```

5769SS1 V4R4M0 990521          AS/400 DUMP          161746/LSVALGAARD/QPADEV0019  06/12/00  9:45:03
DMPSYSOBJ PARAMETERS
OBJ- QWCBT_JOB_INDEX          CONTEXT- QSYS
TYPE- *ALL SUBTYPE-*ALL
OBJECT TYPE- INDEX
NAME- QWCBT_JOB_INDEX          TYPE- 0E SUBTYPE- A4
LIBRARY- QSYS                  TYPE- 04 SUBTYPE- 01
CREATION- 05/13/00 01:38:47    SIZE- 00000E2000
OWNER- QSYS                     TYPE- 08 SUBTYPE- 01
ATTRIBUTES- 0800               ADDRESS- 2E8B5BEEB8 000000
INDEX ATTRIBUTES-
000000 00FF0000 00000071 0EA4D8E6 C3C2E36D D1D6C26D C9D5C4C5 E7404040 40404040 * 0 É uQWCBT_JOB_INDEX *
000020 40404040 40404040 A0040000 00000000 00000000 00000000 00000000 00000000 * µ *
000040 00000000 00000000 0F7056C0 A9000400 00000000 00000000 00000000 00000000 * øi{z *
000060 34003000 20000099 AE000087 DA0015B1 9E * rþ g' fÆ *

```

Looking at the header of the index, we can determine that each entry is evidently 48 bytes long (**0030**) and the key length for the entry is 32 bytes (**0020**).

## Format of the Job Index Entry

In the same dump output of the job index object, the current entries in the index are also shown:

```
INDEX ENTRIES-
. 000001-
000000 F17B7BF0 F0F0F3D7 C64040D4 C4C1D3E8 40404040 40F1F1F4 F8F5F600 00000000 *1##0003PF MDALY 114856 *
000020 80000000 00000000 1D5CC0F2 4B0D4400 * *{2. a
..POINTERS-
000020 SPP 19 D0 QWCBT01 04 01 QSYS 000D4300 0000

...
. 001696-
000000 F1D8D7C1 C4C5E5F0 F0F1F8E3 D4E4D9C4 D6C3D240 40F1F6F1 F2F0F000 00000000 *1QPADEV0018TMURDOCK 161200 *
000020 80000000 00000000 1D5CC0F2 4B06AC00 * *{2. B
..POINTERS-
000020 SPP 19 D0 QWCBT01 04 01 QSYS 0006AB00 0000

. 001697-
000000 F1D8D7C1 C4C5E5F0 F0F1F9D3 E2E5C1D3 C7C1C1D9 C4F1F6F1 F7F4F600 00000000 *1QPADEV0019LSVALGAARD161746 *
000020 80000000 00000000 1D5CC0F2 4B120C00 * *{2.
..POINTERS-
000020 SPP 19 D0 QWCBT01 04 01 QSYS 00120B00 0000

. 001698-
000000 F1D8D7C1 C4C5E5F0 F0F1F9E2 D7E4D9D2 C1E8C1E2 E3F1F6F1 F7F3F600 00000000 *1QPADEV0019SPURKAYAST161736 *
000020 80000000 00000000 1D5CC0F2 4B1EEC00 * *{2. O
..POINTERS-
000020 SPP 19 D0 QWCBT01 04 01 QSYS 001EEB00 0000

...
. 002834-
000000 F2D3C6C1 E8D5D3C5 E8C240F1 F5F7F3F5 F4C9E3D4 E2C3D940 40404000 00000000 *2LFAYNLEYB 157354ITMSCR *
000020 80000000 00000000 1D5CC0F2 4B13B800 * *{2. ½
..POINTERS-
000020 SPP 19 D0 QWCBT01 04 01 QSYS 0013B700 0000

. 002835-
000000 F2D3E2E5 C1D3C7C1 C1D9C4F1 F6F1F2F5 F3D8D7C1 C4C5E5F0 F0F1D100 00000000 *2LSVALGAARD161253QPADEV001J *
000020 80000000 00000000 1D5CC0F2 4B137C00 * *{2. @
..POINTERS-
000020 SPP 19 D0 QWCBT01 04 01 QSYS 00137B00 0000

. 002836-
000000 F2D3E2E5 C1D3C7C1 C1D9C4F1 F6F1F7F4 F6D8D7C1 C4C5E5F0 F0F1F900 00000000 *2LSVALGAARD161746QPADEV0019 *
000020 80000000 00000000 1D5CC0F2 4B120C00 * *{2.
..POINTERS-
000020 SPP 19 D0 QWCBT01 04 01 QSYS 00120B00 0000

. 002837-
000000 F2D3E9C1 E8C4C5E2 404040F0 F8F8F1F1 F1D8D7C1 C4C5E5F0 F0F0D100 00000000 *2LZAYDES 088111QPADEV000J *
000020 80000000 00000000 1D5CC0F2 4B056C00 * *{2. %
..POINTERS-
000020 SPP 19 D0 QWCBT01 04 01 QSYS 00056B00 0000

...
. 004564-
000000 F2E9C7D6 D9C5D3C9 D24040F1 F5F1F7F9 F8D8D7C1 C4C5E5F0 F0F0C600 00000000 *2ZGORELIK 151798QPADEV000F *
000020 80000000 00000000 1D5CC0F2 4B1D9000 * *{2. '
..POINTERS-
000020 SPP 19 D0 QWCBT01 04 01 QSYS 001D8F00 0000
...
```

From the dump, we can determine that each entry has the following format:

```
DCL DD THE-ENTRY CHAR(48) BDRY(16);
      DCL DD ENTRY-PREFIX CHAR( 1) DEF(THE-ENTRY) POS( 1);
      DCL DD ENTRY-JOB-NAME CHAR(26) DEF(THE-ENTRY) POS( 2);
      DCL DD * CHAR( 5) DEF(THE-ENTRY) POS(28);
      DCL SPCPTR .WCB-ENTRY DEF(THE-ENTRY) POS(33);
```

The key is 32 characters long. By inspection of the index entries in the dump we learn that the prefix is “1” if the key is [prefix, device, user, job-number] and “2” if the key is [prefix, user, job-number, device]. The .WCB-ENTRY is a space pointer to the entry in the appropriate WCB Table describing the job.

## Addressing the Job Index

There are two ways you can obtain a pointer to the job index. You can either resolve a pointer to the object:

```
RESOLVE-TO-JOB-INDEX:
  CPYBLA RESOLVE-TYPE, X'0EA4';
  CPYBLAP RESOLVE-NAME, "QWCBT_JOB_INDEX", " ";
  RSLVSP .JOB-INDEX, RESOLVE, *, *;
```

Or you can use the pointer found in **QWCBT00**:

```
DCL DD PCO CHAR(512) BASPCO;
      DCL SYSPTR @QWCBT00 DEF(PCO) POS(433);

DCL DD WCB-ROOT CHAR(2048) BAS(.WCB-ROOT);
      DCL SYSPTR .JOB-INDEX DEF(WCB-ROOT) POS(545); /* offset 220 */

      SETSPFPF .WCB-ROOT, @QWCBT00;
```

Resolving the pointer requires that you have authority to the object if you want to use the pointer or that you set the authority as the fourth operand of the RSLVSP instruction. The pointer found in **QWCBT00** already has the necessary authority set in the pointer, so we shall simply use the pointer as found:

```
000320 00008000 00000000 2E8B5BEE B8000E3F    ..0.....»$0½... Index
```

This means that we do not need any special or specific authority to access the object.

## Finding Entries in the Job Index

The Find Independent Index Entry (**FNDINXEN**) instruction can find entries based on a partial key (viz. the device part of the job name):

```
FNDINXEN .JOB-RECEIVER, .JOB-INDEX, .OPTIONS, .ARGUMENT;
```

The **RULE** option identifies the type of search to be performed. The value of x'0001' means 'Find equal occurrences of the search argument'. We set the **LENGTH** of the argument to 11 to include the prefix and the 10-character device name:

```
DCL SPCPTR .OPTIONS INIT(OPTIONS);
DCL DD     OPTIONS CHAR(10);
DCL DD     OPT-RULE      CHAR(2) DEF(OPTIONS) POS(1) INIT(X'0001');
DCL DD     OPT-ARG-LENGTH BIN(2) DEF(OPTIONS) POS(3) INIT(11);
DCL DD     OPT-ARG-OFFSET BIN(2) DEF(OPTIONS) POS(5) INIT( 0);
DCL DD     OPT-MAX-COUNT  BIN(2) DEF(OPTIONS) POS(7) INIT(1000);
DCL DD     OPT-RETURN-COUNT BIN(2) DEF(OPTIONS) POS(9);
```

Since many jobs associated with any given device could still be in the system we generously allocate room for 1000 such jobs. The code could be enhanced to handle a variable number of jobs, but this hardly seems necessary. Setting the argument is straightforward:

```
DCL SPCPTR .ARGUMENT INIT(ARGUMENT);
DCL DD     ARGUMENT CHAR(32);
DCL DD     ARG-PREFIX      CHAR( 1) DEF(ARGUMENT) POS( 1);
DCL DD     ARG-DEVICE      CHAR(10) DEF(ARGUMENT) POS( 2);
DCL DD     *                CHAR(21) DEF(ARGUMENT) POS(12);

FIND-JOBS-WITH-GIVEN-DEVICE:
  CPYBLA     ARG-PREFIX, "1";
  CPYBLA     ARG-DEVICE, PARM-DEVICE;
  FNDINXEN   .JOB-RECEIVER, .JOB-INDEX, .OPTIONS, .ARGUMENT;
```

## A Faster Version, **MIWCBFND**

Assembling the pieces of code into a complete program (**MIWCBFND**) yields the code below.

```
/* CALL PGM(MIWCBFND) PARM(DEVICE JOBNAM) */
/* INPUT:  DEVICE CHAR(10)  DEVICE NAME FOR TERMINAL */
/* OUTPUT: JOBNAM CHAR(26)  ACTIVE JOB USING THE DEVICE */
/* IF NO JOB IS USING THE DEVICE, RETURNS SPACES */

DCL DD PCO CHAR(512) BASPCO;
DCL SYSPTR @QWCBT00 DEF(PCO) POS(433);

DCL SPCPTR .WCB-ROOT;
DCL DD WCB-ROOT CHAR(2048) BAS(.WCB-ROOT);
DCL SYSPTR .JOB-INDEX DEF(WCB-ROOT) POS(545); /* OFFSET 220 */

DCL SPCPTR .JOB-RECEIVER INIT(JOB-RECEIVER);
DCL DD     JOB-RECEIVER CHAR(48000) BDRY(16);
DCL DD     JOB-ENTRY(1000) CHAR(48) DEF(JOB-RECEIVER) POS(1);

DCL DD THE-ENTRY CHAR(48) BDRY(16);
DCL DD ENTRY-PREFIX      CHAR( 1) DEF(THE-ENTRY) POS( 1);
DCL DD ENTRY-JOB-NAME    CHAR(26) DEF(THE-ENTRY) POS( 2);
DCL DD *                  CHAR( 5) DEF(THE-ENTRY) POS(28);
DCL SPCPTR .WCB-ENTRY      DEF(THE-ENTRY) POS(33);

DCL SPCPTR .ARGUMENT INIT(ARGUMENT);
DCL DD     ARGUMENT CHAR(32);
DCL DD     ARG-PREFIX      CHAR( 1) DEF(ARGUMENT) POS( 1);
DCL DD     ARG-DEVICE      CHAR(10) DEF(ARGUMENT) POS( 2);
DCL DD     *                CHAR(21) DEF(ARGUMENT) POS(12);
```

```

DCL DD WCB-ENTRY CHAR(1024) BAS(.WCB-ENTRY);
DCL DD WCB-JOB CHAR(26) DEF(WCB-ENTRY) POS( 1);
DCL DD WCB-DEVICE CHAR(10) DEF(WCB-JOB) POS( 1);
DCL DD WCB-USER CHAR(10) DEF(WCB-JOB) POS( 11);
DCL DD WCB-NUMBER CHAR( 6) DEF(WCB-JOB) POS( 21);
DCL SYSPTR .WCB-PCS DEF(WCB-ENTRY) POS( 33);
DCL DD WCB-TYPE CHAR( 1) DEF(WCB-ENTRY) POS( 97);
DCL DD WCB-STATUS CHAR( 1) DEF(WCB-ENTRY) POS( 98);
DCL DD WCB-GROUP CHAR( 1) DEF(WCB-ENTRY) POS(100);

DCL SPCPTR .WCB-PCO;
DCL DD WCB-PCO CHAR(992) BAS(.WCB-PCO);
DCL DD WCB-PCO-GROUP-STATUS CHAR(1) DEF(WCB-PCO) POS(548);
DCL DD WCB-PCO-SYSREQ-STATUS CHAR(1) DEF(WCB-PCO) POS(980);

DCL DD ENTRY-NBR BIN(2);

DCL SPCPTR .OPTIONS INIT(OPTIONS);
DCL DD OPTIONS CHAR(4010);
DCL DD OPT-RULE CHAR(2) DEF(OPTIONS) POS(1) INIT(x'0001');
DCL DD OPT-ARG-LENGTH BIN(2) DEF(OPTIONS) POS(3) INIT(11);
DCL DD OPT-ARG-OFFSET BIN(2) DEF(OPTIONS) POS(5) INIT( 0);
DCL DD OPT-MAX-COUNT BIN(2) DEF(OPTIONS) POS(7) INIT(1000);
DCL DD OPT-RETURN-COUNT BIN(2) DEF(OPTIONS) POS(9);
DCL DD OPT-ENTRY1000 CHAR(4) DEF(OPTIONS) POS(11);

DCL SPCPTR .PARAM1 PARM;
DCL DD PARM-DEVICE CHAR(10) BAS(.PARAM1);

DCL SPCPTR .PARAM2 PARM;
DCL DD PARM-JOB CHAR(26) BAS(.PARAM2);
DCL DD PARM-JOB-DEVICE CHAR(10) DEF(PARM-JOB) POS( 1);
DCL DD PARM-JOB-USER CHAR(10) DEF(PARM-JOB) POS(11);
DCL DD PARM-JOB-NBR CHAR( 6) DEF(PARM-JOB) POS(21);

DCL OL PARAMETERS(.PARAM1, .PARAM2) EXT PARM MIN(2);

/*****/

ENTRY * (PARAMETERS) EXT;
CPYBREP PARM-JOB, " ";

GET-JOB-INDEX:
SETSPFP .WCB-ROOT, @QWCBT00;

FIND-JOBS-WITH-GIVEN-DEVICE:
CPYBLA ARG-PREFIX, "1";
CPYBLA ARG-DEVICE, PARM-DEVICE;
FNDINXEN .JOB-RECEIVER, .JOB-INDEX, .OPTIONS, .ARGUMENT;

CPYNV ENTRY-NBR, 0;
CHECK-NEXT-ENTRY:
ADDN(S) ENTRY-NBR, 1;
CMPNV(B) ENTRY-NBR, OPT-RETURN-COUNT/HI(RETURN);
CPYBWP THE-ENTRY, JOB-ENTRY (ENTRY-NBR);

CHECK-JOB-STATUS:
CMPBLA(B) WCB-STATUS, x'20'/NEQ (CHECK-NEXT-ENTRY); /*ACTIVE */
TSTBUM(B) WCB-GROUP, x'06'/ONES(CHECK-NEXT-ENTRY); /*SUSGRP */
CMPBLA(B) WCB-TYPE, "I"/NEQ (CHECK-NEXT-ENTRY); /*INTERACT*/

CHECK-FOR-SYSTEM-REQUEST:
CMPPTRT(B) .WCB-PCS, * /EQ(CHECK-NEXT-ENTRY);
SETSPFP .WCB-PCO, .WCB-PCS;
TSTBUM(B) WCB-PCO-SYSREQ-STATUS, x'80'/NZER(CHECK-NEXT-ENTRY);

HAVE-THE-ACTIVE-ENTRY:
CPYBLA PARM-JOB, WCB-JOB;

RETURN:
RTX *;

```

Because the program uses the SETSPFP MI-instruction to access a system domain object, it must be a system-state program to run at security levels above 30. Using the index we get a significant boost in performance. My tests on a 170-based system show that a job is located in typically less than 1 millisecond.

Placing a breakpoint at the RETURN label shows that the program works:

```
====> ADDBKP STMT(return) PGMVAR(('parm-job' ()))
```

```
====> call miwcbfnd parm(qpadev000g x)
```

```
                                Display Breakpoint
Statement/Instruction . . . . . : RETURN /0011
Program . . . . . : MIWCBFND
Variable . . . . . : PARM-JOB
Type . . . . . : CHARACTER
Length . . . . . : 26
'QPADEV000GLSVALGAARD218889'
```

## ***A Brief History of WCBT Problems***

Until V3R1 the limit for number of entries in the WCBT was 32,767. Back in V2R3, V3R0M5, and V3R1 a PTF came out that would give a CPI1468 warning message when you were running out of WCBT space. Until then you just got a CPF0957 when it was already too late. That PTF was of more importance for V3R0M5 and earlier, as the size of the WCBT was greatly increased at V3R1 - which then caused the following problems.

During the upgrade to V3R1, the WCBT format was converted, and the table was recreated - on a single DASD unit. In V3R1 there were a couple of PTFs to solve the performance problem when the WCBT resided on a single disk unit (it allowed it to be allocated across multiple disk units). A tool (program **QWCWCBTS**) was made available by V3R1 PTF that you could run in a restricted state to reallocate the WCBT over multiple DASD arms. Another PTF in V3R1 addressed the poor performance of commands like **WRKJOB**, **WRKSBSJOB**, and **WRKSBMJOB** due to the WCBT not being compressed during an IPL (this PTF introduced the **QWCBTCMPTB** data area to control compression during IPL, which was replaced at V4R1 by a **CPRJOBTL** attribute set with **CHGIPLA**).

Another V3R1 problem was that, if you had an FSIOP, the **QWCBTCLNUP** job would occasionally still be running when the FSIOP monitor job was started, resulting in a message like CPF1101 or CPD2638. V3R1, V3R2, and V3R6 PTFs were released for that.

If you ran an \*M36 and had an incompletely created one, there was a problem (from V3R7 on) where the M36 would fail to start after the WCBT was compressed (because the pointer to the QM36000 job structure was invalidated by the compression). I believe a PTF was only released for that in V4R4.

Another problem many people encountered, that ate up your disk space at a rapid pace, was the SRDS (Save Restore Descriptor Space) would grow at a rate of 20MB or more per day (under certain conditions - assuming you ran a daily backup). This was originally suspected as a WCBT problem, but a compress of the WCBT in this case didn't help - you needed to run **RCLSTG** frequently. This problem occurred on V3R6, V3R7, V4R1, V4R2, and V4R3. I believe that all but V3R6 got a PTF to fix this.

This historical perspective has been contributed by Neil Palmer at DPS Data Processing Services Canada Ltd.



Blank

## Chapter 10

### Internal Sorting, *Combsort*

#### **Sorting Internal Data**

In this chapter we shall use MI to develop a very fast sorting program for internal data held in an array (a *table*) and show how to get the CPU time the processor spends sorting tables of different sizes. In doing this we'll have occasion to use floating point variables computing logarithms. We start with a description of the simple, well-known Bubble sort algorithm and show the algorithm first in COBOL.

#### **Bubble Sort**

As the name suggests, bubble sort moves items up a table like bubbles in a tube. The algorithm can be explained as follows: pass over the data, comparing and exchanging items so that the largest item ends up at the end of the table. Repeat for the remaining items until the table is sorted, that is, no exchanges were made during the latest pass.

The following COBOL code performs a bubble sort:

```
01  TABLE-TO-SORT.
02  TABLE-SIZE          PIC S9(5)  COMP.
02  TABLE-MAX          PIC S9(5)  COMP VALUE +1000.
02  TABLE-ITEM          OCCURS 1000 TIMES.
03  TABLE-KEY          PIC X(9).
03  TABLE-DATA         PIC X(11).

01  VARIOUS-INDICES.
02  ITEM-NBR            PIC S9(5)  COMP.
02  SWAP-NBR            PIC S9(5)  COMP.
02  JUMP-SIZE           PIC S9(5)  COMP.
02  UPPER-LIMIT        PIC S9(5)  COMP.

01  VARIOUS-VALUES.
02  SWAP-ITEM           PIC X(20).
02  SWAP-INDICATOR      PIC X(1).
88  NO-MORE-SWAPS       VALUE IS SPACE.

BUBBLE-SORT-THE-ARRAY.
  MOVE  nnnn  TO TABLE-SIZE
  MOVE "SWAP" TO SWAP-INDICATOR
  MOVE  1    TO JUMP-SIZE
  PERFORM BUBBLE-SORT
  UNTIL NO-MORE-SWAPS
  .

BUBBLE-SORT.
  MOVE SPACE TO SWAP-INDICATOR
  COMPUTE UPPER-LIMIT = TABLE-SIZE - JUMP-SIZE

  PERFORM COMPARE-AND-SWAP-KEYS
  VARYING ITEM-NBR FROM 1 BY 1
  UNTIL ITEM-NBR > UPPER-LIMIT
  .

COMPARE-AND-SWAP-KEYS.
  COMPUTE SWAP-NBR = ITEM-NBR + JUMP-SIZE
  IF TABLE-KEY (ITEM-NBR) > TABLE-KEY (SWAP-NBR)
    MOVE TABLE-ITEM (ITEM-NBR) TO SWAP-ITEM
    MOVE TABLE-ITEM (SWAP-NBR) TO TABLE-ITEM (ITEM-NBR)
    MOVE SWAP-ITEM TO TABLE-ITEM (SWAP-NBR)
    MOVE "SWAP" TO SWAP-INDICATOR
  .
```

The code sorts a table of a given size assuming that it has already been initialized with values. Even if COBOL is not your favorite language, the code should be easy enough to follow. Study the above code so you understand how it works. It has long been known that bubble sort is the *worst* sorting algorithm and that one should never use it for anything more than about 20 values. The best general-purpose sorting algorithm was known to be quicksort. Bubble sort's only redeeming feature is its simplicity. Sorting is a problem that was solved years ago.

## Combsort

Just as we thought that the last word had been said about sorting, a breakthrough comes along and spoils everything. In the April 1991 issue of BYTE magazine, Stephen Lacey and Richard Box show that a simple modification to bubble sort makes it a fast and efficient sort method on par with heapsort and quicksort.

In a bubble sort, each item is compared to the next; if the two are out of order, they are swapped. This method is slow because it is susceptible to the appearance of what Box and Lacey call *turtles*. A turtle is a relatively low value located near the end of the table. During a bubble sort, this element moves only one position for each pass, so a single turtle can cause maximal slowing. Almost every long table of items contains a turtle.

Their simple modification of bubble sort, which they call ‘combsort’, eliminates turtles quickly by allowing the distance between compared items to be greater than one. This distance - the JUMP-SIZE - is initially set to the TABLE-SIZE. Before each pass, the JUMP-SIZE is divided by 1.3 (the *shrink factor*). If this causes it to become less than 1, it is simply set to 1, collapsing combsort into bubble sort. An exchange of items moves items by JUMP-SIZE positions rather than only one position, causing turtles to jump rather than crawl. As with any sort method where the displacement of an element can be larger than one position, combsort is not stable - like elements do not keep their relative positions. This is rarely a problem in practice and could, if necessary, be fixed by adding a sequence number to the key.

Successively shrinking the JUMP-SIZE is analogous to combing long, tangled hair - stroking first with your fingers alone, then with a pick comb that has widely spaced teeth, followed by finer combs with progressively closer teeth - hence the name **combsort**. Lacey and Box came up with a shrink factor of 1.3 empirically by testing combsort on over 200,000 random tables. There is at present no theoretical justification for this particular value; it just works...

Here is then the magic code. It is clearly correct, as it (unless the table is empty) ends with JUMP-SIZE = 1 (ensured by the ‘+3’) and therefore degenerates into bubble sort:

```
COMBSORT-THE-ARRAY.
  MOVE TABLE-SIZE TO JUMP-SIZE
  PERFORM COMBSORT
  UNTIL NO-MORE-SWAPS
    AND JUMP-SIZE NOT > 1
.

COMBSORT.
  COMPUTE JUMP-SIZE = (10 * JUMP-SIZE + 3) / 13
  COMPUTE UPPER-LIMIT = TABLE-SIZE - JUMP-SIZE
  MOVE SPACE TO SWAP-INDICATOR
  PERFORM COMPARE-AND-SWAP-KEYS
    VARYING ITEM-NBR FROM 1 BY 1
    UNTIL ITEM-NBR > UPPER-LIMIT
.
```

The careful termination test (JUMP-SIZE NOT > 1) also caters for the case where the table is empty.

## MI-Version of Combsort

We’ll make the MI-version of Combsort, **MICMBRSRT**, a separate program that we can call from our test program. This nicely separates the sorting algorithm from the test scaffolding. For simplicity, we make the key the same as the table element value. Then we generalize a bit and make the code able to handle both ascending keys and descending keys. The first parameter is a template, that gives the number of elements to be sorted and the sort direction:

```
DCL SPCPTR .PARM1 PARM;
DCL DD PARM1 CHAR(5) BAS(.PARM1);
  DCL DD PARM-NBR-ELEMENTS PKD(7,0) DEF(PARM1) POS(1);
  DCL DD PARM-DIRECTION CHAR(1) DEF(PARM1) POS(5); /* A or D */

DCL SPCPTR .PARM2 PARM;
DCL DD ELEMENT(1) CHAR(10) BAS(.PARM2);
DCL DD KEY (1) CHAR(10) BAS(.PARM2); /* overlays ELEMENT */

DCL OL PARMS(.PARM1, .PARM2) EXT PARM MIN(2);
```

We do not want to know how large the table is, so it is declared here to contain only one element. We’ll tell the compiler to omit subscript checking. This is done with the “Override Program Attributes”-instruction

(**OVRPGATR**), which really is executed at compile-time and is in effect until overridden by another **OVRPGATR**.

## Override Program Attribute

The format is: **OVRPGATR**      *Attribute ID, Attribute modifier*

Attribute ID	Description	Modifier
<b>1</b>	Array constraintment	1 = Constrain array references <b>2</b> = Do not constrain array references 3 = Fully unconstrain array references 4 = Resume attribute given in template
2	String constraintment	1 = Constrain string references 2 = Do not constrain string references 3 = Resume attribute given in template
3	Suppress binary size error	1 = Suppress binary size exceptions 2 = Do not suppress binary size exceptions 3 = Resume attribute given in template
4	Suppress decimal data error	1 = Suppress decimal data exceptions 2 = Do not suppress decimal data exceptions 3 = Resume attribute given in template
5	CPYBWP alignment	1 = Require like alignment 2 = Do not require like alignment
6	CMPSPAD null pointer	1 = Signal pointer does not exist exception 2 = Do not signal pointer does not exist exception

```
ENTRY * (PARMS) EXT;
  OVRPGATR 1, 2; /* Don't Constrain Array Refs */
```

The effect of the **OVRPGATR** in this case allows us to access the array as an *unbounded* array, this allows us to declare the variable as only having one element, ELEMENT(1), but in our code we can use any positive value as the index to access any element within the array. This is a very useful facility, however as the compiler is not now checking whether the index is within the bounds of the array, it can be quite easy to supply an index whose value is so large, that we try to access data outside the storage space for the array. Don't worry though, OS/400 has a polite way of telling you, with a MCH error.

## The Sort Double Loop

Declare the various local variables:

```
DCL DD SWAP-FLAG CHAR(1);
DCL DD JUMP-SIZE BIN(4);
DCL DD SWEEP-END BIN(4);
DCL DD ITEM-NBR BIN(4);
DCL DD COMP-NBR BIN(4);
```

Start the loops:

```
      CPYNV      JUMP-SIZE, PARM-NBR-ELEMENTS;
SORT-JUMP:
  CMPNV(B)      JUMP-SIZE, 1 /HI(SORT-SWEEP);
  CMPBLA(B)     SWAP-FLAG, "S"/NEQ(RETURN);
SORT-SWEEP:
  MULT(S)       JUMP-SIZE, 10;
  ADDN(S)        JUMP-SIZE, 3;
  DIV(S)         JUMP-SIZE, 13; /* JUMP-SIZE = (10 * JUMP-SIZE + 3)/13 */
  SUBN          SWEEP-END, PARM-NBR-ELEMENTS, JUMP-SIZE;
  CPYBLA        SWAP-FLAG, " ";
  CPYNV(B)      ITEM-NBR, 0/ZER(SORT-COMPARE);

SORT-SWAP:
  EXCHBY        ELEMENT(ITEM-NBR), ELEMENT(COMP-NBR);
  CPYBLA        SWAP-FLAG, "S";
SORT-COMPARE:
  ADDN(S)        ITEM-NBR, 1;
  CMPNV(B)       ITEM-NBR, SWEEP-END/HI(SORT-JUMP);
  ADDN           COMP-NBR, ITEM-NBR, JUMP-SIZE;
  CMPBLA(B)      PARM-DIRECTION, "D"/EQ(DESCENDING-SORT-COMPARE);

ASCENDING-SORT-COMPARE:
  CMPBLA(B)      KEY(ITEM-NBR), KEY(COMP-NBR)/
  HI(SORT-SWAP), NHI(SORT-COMPARE);
```

```

DESCENDING-SORT-COMPARE:
  CMPBLA(B)  KEY(ITEM-NBR), KEY(COMP-NBR)/
              LO(SORT-SWAP), NLO(SORT-COMPARE);
RETURN:
  RTX      *;

```

Note the handy swap instruction: **EXCHBY** (Exchange Bytes). Note also the use of more than one branch extender:

```

      CMPBLA(B)  KEY(ITEM-NBR), KEY(COMP-NBR)/      /* if condition is 'HI' goto SORT-SWAP */
              HI(SORT-SWAP), NHI(SORT-COMPARE);/* else goto SORT-COMPARE */

```

Since the second condition is the negation of the first, a branch is always taken.

## Complete Combsort Code

```

DCL SPCPTR .PARM1 PARM;
DCL DD PARM1 CHAR(5) BAS(.PARM1);
DCL DD PARM-NBR-ELEMENTS PKD(7,0) DEF(PARM1) POS(1);
DCL DD PARM-DIRECTION CHAR(1) DEF(PARM1) POS(5); /* A OR D */

DCL SPCPTR .PARM2 PARM;
DCL DD ELEMENT(1) CHAR(10) BAS(.PARM2);
DCL DD KEY (1) CHAR(10) BAS(.PARM2);

DCL OL PARMS(.PARM1, .PARM2) EXT PARM MIN(2);

DCL DD SWAP-FLAG CHAR(1);
DCL DD JUMP-SIZE BIN(4);
DCL DD SWEEP-END BIN(4);
DCL DD ITEM-NBR BIN(4);
DCL DD COMP-NBR BIN(4);

ENTRY * (PARMS) EXT;
  OVRPGATR 1, 2; /* DON'T CONSTRAIN ARRAY REFS */
  CPYV JUMP-SIZE, PARM-NBR-ELEMENTS;
SORT-JUMP:
  CMPNV(B) JUMP-SIZE, 1 /HI(SORT-SWEEP);
  CMPBLA(B) SWAP-FLAG, "S"/NEQ(RETURN);
SORT-SWEEP:
  MULT(S) JUMP-SIZE, 10;
  ADDN(S) JUMP-SIZE, 3;
  DIV(S) JUMP-SIZE, 13;
  SUBN SWEEP-END, PARM-NBR-ELEMENTS, JUMP-SIZE;
  CPYBLA SWAP-FLAG, " ";
  CPYV(B) ITEM-NBR, 0/ZER(SORT-COMPARE);

SORT-SWAP:
  EXCHBY ELEMENT(ITEM-NBR), ELEMENT(COMP-NBR);
  CPYBLA SWAP-FLAG, "S";
SORT-COMPARE:
  ADDN(S) ITEM-NBR, 1;
  CMPNV(B) ITEM-NBR, SWEEP-END/HI(SORT-JUMP);
  ADDN COMP-NBR, ITEM-NBR, JUMP-SIZE;
  CMPBLA(B) PARM-DIRECTION, "D"/EQ(DESCENDING-SORT-COMPARE);

ASCENDING-SORT-COMPARE:
  CMPBLA(B) KEY(ITEM-NBR), KEY(COMP-NBR)/
              HI(SORT-SWAP), NHI(SORT-COMPARE);
DESCENDING-SORT-COMPARE:
  CMPBLA(B) KEY(ITEM-NBR), KEY(COMP-NBR)/
              LO(SORT-SWAP), NLO(SORT-COMPARE);

RETURN:
  RTX      *;

```

## Using Instruction Pointers

The test of the sort direction in the inner loop can be avoided using an instruction pointer with the “Set Instruction Pointer”-instruction, **SETIP** *pointer, label*:

```

DCL INSPTR KEY-COMPARE;

ENTRY * (PARMS) EXT;
  OVRPGATR 1, 2; /* DON'T CONSTRAIN ARRAY REFS */
  SETIP KEY-COMPARE, ASCENDING-SORT-COMPARE;

```

```

CMPBLA(B)  PARM-DIRECTION, "D"/NEQ(=+2);
SETIP      KEY-COMPARE, DESCENDING-SORT-COMPARE;;

...
SORT-COMPARE:
ADDN(S)    ITEM-NBR, 1;
CMPNV(B)   ITEM-NBR, SWEEP-END/HI(SORT-JUMP);
ADDN(B)    COMP-NBR, ITEM-NBR, JUMP-SIZE/POS(KEY-COMPARE);

ASCENDING-SORT-COMPARE:
CMPBLA(B)  KEY(ITEM-NBR), KEY(COMP-NBR)/
           HI(SORT-SWAP), NHI(SORT-COMPARE);
DESCENDING-SORT-COMPARE:
CMPBLA(B)  KEY(ITEM-NBR), KEY(COMP-NBR)/
           LO(SORT-SWAP), NLO(SORT-COMPARE);

```

## Testing Combsort

The test program (**MITSTCBM**) shall run through a range of table sizes, say from 50,000 items to 1,000,000 items. For each choice of table size,  $N$ , we initialize the table with random numbers (giving us a chance of showing how to do that too), then get the processor time spent before and after calling the Combsort program, **MICBSRT**. Finally, we calculate the time difference and compare it to the theoretical optimal value  $kN \log_2 N$ , where  $k$  is a constant (the time to decide if a given item is in place and move it if not).

First the interface to Combsort:

```

DCL SPCPTR .CONTROL INIT(CONTROL);
DCL DD     CONTROL CHAR(5);
           DCL DD     CTRL-NBR-ELEMENTS PKD(7,0) DEF(CONTROL) POS(1);
           DCL DD     CTRL-DIRECTION   CHAR(1)  DEF(CONTROL) POS(5);

DCL SPCPTR .TABLE INIT(TABLE);
DCL DD     TABLE(1000000) ZND(10,0); /* MAX 16MB */

DCL SYSPTR .COMBSORT INIT("MICBSRT", TYPE(PGM));
DCL OL COMBSORT(.CONTROL, .TABLE) ARG;

```

## Generating Random Numbers

We use the linear congruential method to generate pseudo-random numbers. The following formula generates evenly distributed integers between 0 and 2,099,862 and suits us fine (seed *Random* with a suitable number first):

$$Random = (1005973 * Random + 443771) \bmod 2099863$$

```

DCL DD RND-RESULT      PKD( 7,0); /* Also SEED value */
DCL DD RND-PRODUCT     PKD(15,0);
DCL DD RND-MODULUS     PKD( 7,0) INIT(P'+2099863');
DCL DD RND-MULTIPLIER  PKD( 7,0) INIT(P'+1005973');
DCL DD RND-INCREMENT   PKD( 7,0) INIT(P'+443771');

DCL DD NBR             BIN(4);
DCL DD N               BIN(4); /* Number of elements in the table */

           CPYNV        RND-RESULT, 314159; /* SEED value */
           CPYNV        NBR, 0;

NEXT-ELEMENT:
MULT       RND-PRODUCT, RND-MULTIPLIER, RND-RESULT;
ADDN(S)    RND-PRODUCT, RND-INCREMENT;
REM        RND-RESULT , RND-PRODUCT, RND-MODULUS; /* REMainder */

ADDN(S)    NBR, 1;
CPYNV     TABLE(NBR), RND-RESULT;
CMPNV(B)   NBR, N/LO(NEXT-ELEMENT);

```

## Measuring Processor Time Spent

The “Materialize Process Attributes”-instruction (**MATPRATR**) can return all kinds of data about a running process. If called with a null 2<sup>nd</sup> operand it returns information about the current process (the one your program is running in): The 3<sup>rd</sup> operand selects what information to materialize; x’21’ selects the processor time spent:

```

DCL SPCPTR .MAT-PROC-ATTRS INIT(MAT-PROC-ATTRS);
DCL DD MAT-PROC-ATTRS CHAR(22);
DCL DD MAT-BYTES-PROVIDED BIN(4) DEF(MAT-PROC-ATTRS) POS( 1);
DCL DD MAT-BYTES-AVAILABLE BIN(4) DEF(MAT-PROC-ATTRS) POS( 5);
DCL DD MAT-TOTAL-BYTES-USED BIN(4) DEF(MAT-PROC-ATTRS) POS( 9);
DCL DD MAT-CPU-TIME-USED CHAR(8) DEF(MAT-PROC-ATTRS) POS(13);
DCL DD MAT-NBR-OF-LOCKS BIN(2) DEF(MAT-PROC-ATTRS) POS(21);

CPYNV MAT-BYTES-PROVIDED, 20; /* don't care about the LOCKS */
MATPRATR .MAT-PROC-ATTRS, *, X'21'; /* get CPU-time used */

```

Store the 8-character timestamp-format CPU-TIME-USED in BEFORE and AFTER variables:

```

DCL DD CPU-TIMES CHAR(24) BDRY(8);
DCL DD CPU-BEFORE CHAR(8) DEF(CPU-TIMES) POS( 1);
DCL DD CPU-AFTER CHAR(8) DEF(CPU-TIMES) POS( 9);
DCL DD CPU-DIFFERENCE CHAR(8) DEF(CPU-TIMES) POS(17);

CPYBLA CPU-BEFORE, MAT-CPU-TIME-USED;

```

## Calling Combsort

```

SORT:
CPYNV CTRL-NBR-ELEMENTS, N;
CPYBLA CTRL-DIRECTION, "ASCENDING"; /* only stores the 'A' */
BRK "1";

CPYNV MAT-BYTES-PROVIDED, 20;
MATPRATR .MAT-PROC-ATTRS, *, X'21';
CPYBLA CPU-BEFORE, MAT-CPU-TIME-USED;

CALLX .COMBSORT, COMBSORT, *;

MATPRATR .MAT-PROC-ATTRS, *, X'21';
CPYBLA CPU-AFTER, MAT-CPU-TIME-USED;
BRK "2";

```

The two break points are placed such that you can observe the table before and after sorting it. We should also test if the table *is* actually sorted by comparing each item with the previous one:

```

DCL DD PREV BIN(4);

TEST-IF-SORTED:
CPYNV NBR, N;
COMPARE-WITH-PREVIOUS:
SUBN(B) PREV, NBR, 1/NPOS(=+3);
CMPBLA(B) TABLE(NBR), TABLE(PREV)/LO(NOT-SORTED); /* Notify if bad */
SUBN(SB) NBR, 1/POS(COMPARE-WITH-PREVIOUS);

```

## Computing $N \log_2 N$

The machine has a built-in function to calculate the natural logarithm (to base  $e$ ) of a number. The logarithm to base 2 is readily found from:  $\ln(x) = \ln(2^{\log_2 x}) = \log_2(x) \ln(2)$ , i.e.:  $\log_2(x) = \ln(x) / \ln(2)$ . All these calculations must be done in floating-point:

```

DCL DD LN-N FLT(8);
DCL DD LOG2-N FLT(8);
DCL DD LN-2 FLT(8);
DCL DD FLT-N FLT(8);
DCL DD N*LOG2-N FLT(8);
DCL DD RESULT ZND(10,0);

COMPUTE-N*LOG2-N:
CPYNV RESULT, N;
CPYBLAP MSG-TEXT, RESULT, " "; /* MSG-TEXT = N */

CPYNV FLT-N, N;
CMF1 LN-N, X'0011', FLT-N;
CMF1 LN-2, X'0011', E'2';
DIV LOG2-N, LN-N, LN-2;
MULT N*LOG2-N, N, LOG2-N;
CPYNV(R) RESULT, N*LOG2-N;
CPYBLA MSG-TEXT(13:10), RESULT; /* MSG-TEXT append N*log2 N */

```

## Compute Mathematical Function with 1 Argument

The **CMF1**-instruction has this format:

**CMF1**                      *Result, 2-Character Function Code, Input Argument*

The *Result* and the *Argument* must both be *floating-point* numbers. This is a (sad) departure from the polymorphic nature of the other instructions that work on numeric operands. So, we need to convert N ourselves:

```
CPYNV                      FLT-N, N;    /* converting to floating point */  
CMF1                      LN-N, X'0011', FLT-N;
```

Function x'0011' is the Natural Logarithm. Other functions are:

x'0001'	Sine	$-1 \leq \sin(x) \leq +1$
x'0002'	Arc sine	$-\pi/2 \leq \arcsin(x) \leq +\pi/2$
x'0003'	Cosine	$-1 \leq \cos(x) \leq +1$
x'0004'	Arc cosine	$0 \leq \arccos(x) \leq \pi$
x'0005'	Tangent	$-\infty \leq \tan(x) \leq +\infty$
x'0006'	Arc tangent	$-\pi/2 \leq \arctan(x) \leq +\pi/2$
x'0007'	Cotangent	$-\infty \leq \cot(x) \leq +\infty$
x'0010'	Exponential function	$0 \leq \exp(x) \leq +\infty$
x'0011'	Natural logarithm	$-\infty \leq \ln(x) \leq +\infty$
x'0012'	Sine hyperbolic	$-\infty \leq \sinh(x) \leq +\infty$
x'0013'	Cosine hyperbolic	$+1 \leq \cosh(x) \leq +\infty$
x'0014'	Tangent hyperbolic	$-1 \leq \tanh(x) \leq +1$
x'0015'	Arc tangent hyperbolic	$-\infty \leq \operatorname{arctanh}(x) \leq +\infty$
x'0020'	Square root	$0 \leq \sqrt{x} \leq +\infty$

To calculate the natural logarithm of 2, we can use the long floating-point literal E'2':

```
CMF1                      LN-2, X'0011', E'2';
```

## Computing Time Differences

Working with 64-bit long integers is complicated by there not being a **BIN(8)** variable type. We'll use the same technique as in chapter 2 of going through intermediate packed numbers:

```
DCL DD CPU-TIMES CHAR(24) BDRY(8);  
DCL DD CPU-BEFORE            CHAR(8) DEF(CPU-TIMES) POS( 1);  
DCL DD CPU-AFTER             CHAR(8) DEF(CPU-TIMES) POS( 9);  
DCL DD CPU-DIFFERENCE        CHAR(8) DEF(CPU-TIMES) POS(17);  
  
DCL DD CPU-DIFF-HI            BIN(4) UNSGND DEF(CPU-DIFFERENCE) POS(1);  
DCL DD CPU-DIFF-LO            BIN(4) UNSGND DEF(CPU-DIFFERENCE) POS(5);  
  
DCL DD TIMESTAMP             PKD(21,0); /* Can hold 64-bit unsigned integer */  
DCL DD TIMESTAMP-HI           PKD(11,0); /* Can hold 32-bit unsigned integer */  
DCL DD TIMESTAMP-LO           PKD(11,0); /* Can hold 32-bit unsigned integer */  
DCL DD TWO**32                PKD(11,0) INIT(P'4294967296'); /* unsigned 232 */
```

The 8-character time difference can be computed with the “Subtract Logical Character”-instruction (**SUBLC**):

```
COMPUTE-TIME-USED:  
SUBLC                      CPU-DIFFERENCE, CPU-AFTER, CPU-BEFORE; /* diff = after - before */  
CPYNV                      TIMESTAMP-HI, CPU-DIFF-HI;                /* copy UNSIGNED values */  
CPYNV                      TIMESTAMP-LO, CPU-DIFF-LO;  
MULT                        TIMESTAMP, TIMESTAMP-HI, TWO**32;  
ADDN(S)                     TIMESTAMP, TIMESTAMP-LO; /* complete 64-bit timestamp as PKD */
```

To convert the timestamp to microseconds, we divide by 4096:

```
DIV                          RESULT, TIMESTAMP, 4096; /* MICROSECONDS */  
CPYBLA                      MSG-TEXT(25:10), RESULT; /* Append to message */
```

If the relationship is linear the time taken to sort N items divided by  $N \log_2 N$  should be the constant (k) slope of the line we get by plotting the two quantities against each other. We compute the slope to four decimal places:



```

DCL DD SLOPE      ZND(10,4);

COMPUTE-SLOPE:
  DIV      SLOPE, RESULT, N*LOG2-N;
  CPYBLA    MSG-TEXT(37:10), SLOPE; /* Append to message */

```

## Simple Numeric Editing

The message that we have built so far displays each number as a raw 10-digit zoned value complete with leading zeroes, such as: “0000050000 0000780482 0005101264 0000065360”. In order to make the result a little more pleasing to the eye, let’s remove the leading zeroes. We set up an 11-character substring into the message, identified by the index value `START`. We then let `START` run through the values 37, 25, 13, and 1 (step of 12) and count the leading zeroes, before replacing them with blanks. No rocket science here. How do we count leading zeroes? The “Verify”-instruction (**VERIFY**) comes in handy:

```

VERIFY      where, source, class

```

Each character of the *source* operand is checked to verify that it is among the characters in the *class* operand. If a match exists, the next character is checked. When a mismatch is found its character position in the source operand is returned in the *where* operand. So, if the instruction `VERIFY WHERE, “00001234”, “01”` would return with `WHERE = 6`. If no mismatch is found, the *where* operand is set to zero. `VERIFY WHERE, “00001234”, “0”` would return `WHERE = 5`. The number of leading zeroes is then one less. The code below does the trick. There are some careful additional tests to cater for various end-conditions (no leading zeroes, all zeroes...):

```

DCL DD WHERE BIN(4);
DCL DD START BIN(4);

      CPYNV      START, 37;
EDIT-RESULT:
  VERIFY      WHERE, MSG-TEXT(START:11), "0";
  SUBN(SB)    WHERE, 1/NPOS(=+2);
  CPYBLAP     MSG-TEXT(START:WHERE), " ", " ";
  SUBN(SB)    START, 12/POS(EDIT-RESULT);

```

I would have preferred `CPYBREP MSG-TEXT(START:WHERE), “ ”` for replacing the leading zeroes, but unfortunately `CPYBREP` does not support substrings ☹.

Finally, show the result and continue the loop until `N` is not lower than the 1,000,000 we had selected as our maximum table size:

```

SHOW-RESULT:
  CALLI      SHOW-MESSAGE, *, .SHOW-MESSAGE;
  CMPNV(B)   N, 1000000/LO(NEXT-SIZE);
  RTX        *;

NOT-SORTED:
  CPYBLAP     MSG-TEXT, "Not Sorted!", " ";
  CALLI      SHOW-MESSAGE, *, .SHOW-MESSAGE;
  RTX        *;

%INCLUDE SHOWMSG

```

## The Complete **MITSTCMB** Test Program

Here is the complete **MITSTCMB** test program:

```

DCL SPCPTR .CONTROL INIT(CONTROL);
DCL DD     CONTROL CHAR(5);
      DCL DD CTRL-NBR-ELEMENTS PKD(7,0) DEF(CONTROL) POS(1);
      DCL DD CTRL-DIRECTION   CHAR(1) DEF(CONTROL) POS(5);

DCL SPCPTR .TABLE INIT(TABLE);
DCL DD     TABLE(1000000) ZND(10,0); /* MAX 16MB */

DCL SYSPTR .COMBSORT INIT("MITCMBST", TYPE(PGM));
DCL OL COMBSORT(.CONTROL, .TABLE) ARG;

DCL DD PREV BIN(4);
DCL DD NBR  BIN(4);

```

```

DCL DD N      BIN(4);
DCL DD WHERE  BIN(4);
DCL DD START  BIN(4);

DCL DD LN-N    FLT(8);
DCL DD LOG2-N  FLT(8);
DCL DD LN-2    FLT(8);
DCL DD FLT-N   FLT(8);
DCL DD N*LOG2-N FLT(8);
DCL DD RESULT  ZND(10,0);
DCL DD SLOPE   ZND(10,4);

DCL DD RND-RESULT      PKD( 7,0); /* ALSO SEED VALUE */
DCL DD RND-PRODUCT     PKD(15,0);
DCL DD RND-MODULUS     PKD( 7,0) INIT(P'+2099863');
DCL DD RND-MULTIPLIER   PKD( 7,0) INIT(P'+1005973');
DCL DD RND-INCREMENT    PKD( 7,0) INIT(P'+443771');

DCL DD CPU-TIMES CHAR(24) BDRY(8);
DCL DD CPU-BEFORE   CHAR(8) DEF(CPU-TIMES) POS( 1);
DCL DD CPU-AFTER    CHAR(8) DEF(CPU-TIMES) POS( 9);
DCL DD CPU-DIFFERENCE CHAR(8) DEF(CPU-TIMES) POS(17);

DCL DD CPU-DIFF-HI   BIN(4) UNSGND DEF(CPU-DIFFERENCE) POS(1);
DCL DD CPU-DIFF-LO   BIN(4) UNSGND DEF(CPU-DIFFERENCE) POS(5);

DCL DD TIMESTAMP     PKD(21,0);
DCL DD TIMESTAMP-HI   PKD(11,0);
DCL DD TIMESTAMP-LO   PKD(11,0);
DCL DD TWO**32        PKD(11,0) INIT(P'4294967296');

DCL SPCPTR .MAT-PROC-ATTRS INIT(MAT-PROC-ATTRS);
DCL DD     MAT-PROC-ATTRS CHAR(22);
DCL DD     MAT-BYTES-PROVIDED BIN(4) DEF(MAT-PROC-ATTRS) POS( 1);
DCL DD     MAT-BYTES-AVAILABLE BIN(4) DEF(MAT-PROC-ATTRS) POS( 5);
DCL DD     MAT-TOTAL-BYTES-USED BIN(4) DEF(MAT-PROC-ATTRS) POS( 9);
DCL DD     MAT-CPU-TIME-USED CHAR(8) DEF(MAT-PROC-ATTRS) POS(13);
DCL DD     MAT-NBR-OF-LOCKS BIN(2) DEF(MAT-PROC-ATTRS) POS(21);

ENTRY * EXT;
CPYNV      N, 0;
NEXT-SIZE:
ADDN(S)    N, 50000;
CPYNV      RND-RESULT, 314159;
CPYNV      NBR, 0;
NEXT-ELEMENT:
MULT       RND-PRODUCT, RND-MULTIPLIER, RND-RESULT;
ADDN(S)    RND-PRODUCT, RND-INCREMENT;
REM        RND-RESULT, RND-PRODUCT, RND-MODULUS;

ADDN(S)    NBR, 1;
CPYNV      TABLE(NBR), RND-RESULT;
CMPNV(B)   NBR, N/LO(NEXT-ELEMENT);

SORT:
CPYNV      CTRL-NBR-ELEMENTS, N;
CPYBLA     CTRL-DIRECTION, "ASCENDING";
BRK "1";

CPYNV      MAT-BYTES-PROVIDED, 20;
MATPRATR   .MAT-PROC-ATTRS, *, X'21';
CPYBLA     CPU-BEFORE, MAT-CPU-TIME-USED;

CALLX      .COMBSORT, COMBSORT, *;

MATPRATR   .MAT-PROC-ATTRS, *, X'21';
CPYBLA     CPU-AFTER, MAT-CPU-TIME-USED;
BRK "2";

TEST-IF-SORTED:
CPYNV      NBR, N;
COMPARE-WITH-PREVIOUS:
SUBN(B)    PREV, NBR, 1/NPOS(=+3);
CMPBLA(B)   TABLE(NBR), TABLE(PREV)/LO(NOT-SORTED);
SUBN(SB)    NBR, 1/POS(COMPARE-WITH-PREVIOUS);:

COMPUTE-N*LOG2-N:
CPYNV      RESULT, N;
CPYBLAP    MSG-TEXT, RESULT, " ";

CPYNV      FLT-N, N;

```

```

CMF1      LN-N, X'0011', FLT-N;
CMF1      LN-2, X'0011', E'2';
DIV       LOG2-N, LN-N, LN-2;
MULT      N*LOG2-N, N, LOG2-N;
CPYNV(R)  RESULT, N*LOG2-N;
CPYBLA    MSG-TEXT(13:10), RESULT;

COMPUTE-TIME-USED:
SUBLC     CPU-DIFFERENCE, CPU-AFTER, CPU-BEFORE;
CPYNV     TIMESTAMP-HI, CPU-DIFF-HI;
CPYNV     TIMESTAMP-LO, CPU-DIFF-LO;
MULT      TIMESTAMP, TIMESTAMP-HI, TWO**32;
ADDN(S)   TIMESTAMP, TIMESTAMP-LO;

DIV       RESULT, TIMESTAMP, 4096; /* MICROSECONDS */
CPYBLA    MSG-TEXT(25:10), RESULT;

COMPUTE-SLOPE:
DIV       SLOPE, RESULT, N*LOG2-N;
CPYBLA    MSG-TEXT(37:10), SLOPE;

CPYNV     START, 37;
EDIT-RESULT:
VERIFY    WHERE, MSG-TEXT(START:11), "0";
SUBN(SB)  WHERE, 1/NPOS(=+2);
CPYBLAP   MSG-TEXT(START:WHERE), " ", " ";
SUBN(SB)  START, 12/POS(EDIT-RESULT);

SHOW-RESULT:
CALLI     SHOW-MESSAGE, *, .SHOW-MESSAGE;
CMPNV(B)  N, 1000000/LO(NEXT-SIZE);
RTX       *;

NOT-SORTED:
CPYBLAP   MSG-TEXT, "Not Sorted!", " ";
CALLI     SHOW-MESSAGE, *, .SHOW-MESSAGE;
RTX       *;

%INCLUDE SHOWMSG

```

## Performance Results

Below are the results of running our test program. Amazingly, Combsort seems to be a true ' $kN \log_2 N$ ' sorting algorithm, with the constant  $k$  being 6.725 microseconds on the average (for the 170 box I was running on):

$N$	$N \log_2 N$	Time ( $\mu$ s)	Ratio
50000	780482	5101264	6.5360
100000	1660964	10797200	6.5006
150000	2579190	17325928	6.7176
200000	3521928	23176840	6.5807
250000	4482892	30809992	6.8728
300000	5458381	37077600	6.7928
350000	6445948	42506240	6.5943
400000	7443856	49615200	6.6653
450000	8450804	56946104	6.7385
500000	9465784	63365320	6.6941
550000	10487990	72378584	6.9011
600000	11516762	77620296	6.7398
650000	12551552	85657896	6.8245
700000	13591897	92395368	6.7978
750000	14637398	98955256	6.7604
800000	15687712	103698432	6.6102
850000	16742538	114234688	6.8230
900000	17801609	121012736	6.7979
950000	18864690	127639992	6.7661
1000000	19931569	134387000	6.7424

$$\text{Time} = 6.725 * N * \log_2 N \mu\text{sec}$$

The “optimization” using an instruction pointer in the inner loop is actually slightly slower for an ascending sort because a branch to the instruction pointer is always taken while the original code with compare and branch only actually branched when the test failed. Since a branch is an expensive operation on a pipelined

RISC machine with pre-fetching of instructions, if you can avoid the branch you gain some speed. For a descending sort where the branch must be taken, both versions run at the same speed.

## **RPG Version of *COMBSORT***

Phil Hall has contributed the following RPG/ILE version of 'COMBSORT'. He says:

"I changed the interface slightly, so as to make it more generic. You now pass:

- The number of elements in the table
- The size of each element
- Size of the sort key
- Sort direction
- The table

This allows COMBSORT to cope with different size elements and sort on varying key sizes. It's not very elegant in RPG, but then again what is, but it does the job."

```
H dftactgrp( *no ) bnddir( 'QC2LE' )

*=====
*==== Prototype the external functions we will be calling
*=====
D memcmp          PR          10I 0 Extproc( '__memcmp' )
*- Used to compare data
D #source1         *          value
D #source2         *          value
D #nbytes          10I 0 value

D memmove         PR          *      Extproc( '_MEMMOVE' )
*- Used to get move data
D #target          *          value
D #source          *          value
D #nbytes          10I 0 value

*=====
*==== Prototype the internal functions we will be calling
*=====
D combsort        PR          *
*- returns a pointer to the sorted table
D #nbrElements    7P 0 value
D #sizeElements   10I 0 value
D #sizeKey        10I 0 value
D #sortDir        1A  value
D #table          *          value

*=====
*==== Main line code
*=====

D dummy          S          *
D testArray      S          15A  Dim( 20 ) CData

C                      Eval      dummy = combsort( 10 :
C                      15 :
C                      15 :
C                      'A' :
C                      %Addr( testArray ) )

*-- all done...
C                      Eval      *INLR = *on

*=====
* Procedure-- processDir
*=====
P combsort        B
D combsort        PI          *
*- returns a pointer to the sorted table
D #nbrElements    7P 0 value
D #sizeElement    10I 0 value
D #sizeKey        10I 0 value
D #sortDir        1A  value
D #table          *          value
```

```

*-- misc procedure variables
D swapFlag      S      1A
D jumpSize      S      4B 0
D sweepEnd      S      4B 0
D itemNbr       S      4B 0
D compNbr       S      4B 0
D compTest      S      10I 0
D elmtItemPtr   S      *
D elmtCompPtr   S      *
D keyItemPtr    S      *
D keyCompPtr    S      *
D swapPtr       S      *
D dummyPtr      S      *

*-- begin procedure code

C      Eval      jumpSize = #nbrElements
C      Alloc     #sizeElement swapPtr

C      SORT@JUMP Tag
C      jumpSize  CabGT 1          SORT@SWEEP
C      swapFlag  CabNE 's'      ALL@DONE

C      SORT@SWEEP Tag
C      Eval      jumpSize = ((jumpSize * 10) + 3) / 13
C      Eval      sweepEnd = #nbrElements - jumpSize
C      Eval      swapFlag = ' '
C      Eval      itemNbr = -1
C      itemNbr   CabEQ -1          SORT@COMP

C      SORT@SWAP Tag
C      Eval      dummyPtr = memmove(swapPtr      :
C      Eval      elmtItemPtr :
C      Eval      #sizeElement )
C      Eval      dummyPtr = memmove(elmtItemPtr :
C      Eval      elmtCompPtr :
C      Eval      #sizeElement )
C      Eval      dummyPtr = memmove(elmtCompPtr :
C      Eval      swapPtr      :
C      Eval      #sizeElement )
C      Eval      swapFlag = 's'

C      SORT@COMP Tag
C      Eval      itemNbr = itemNbr + 1
C      itemNbr   CabGT sweepEnd   SORT@JUMP
C      Eval      elmtItemPtr = #table +
C      Eval      (itemNbr * #sizeElement)
C      Eval      keyItemPtr = #table +
C      Eval      (itemNbr * #sizeElement)
C      Eval      compNbr = (itemNbr + jumpSize) - 1
C      Eval      keyCompPtr = #table +
C      Eval      (compNbr * #sizeElement)
C      Eval      elmtCompPtr = #table +
C      Eval      (compNbr * #sizeElement)
C      #sortDir  CabEQ 'd'      DESC@SORT

C      ASCE@SORT Tag
C      Eval      compTest = memcmp( keyItemPtr :
C      Eval      keyCompPtr :
C      Eval      #sizeKey )
C      Select
C      when      compTest > 0
C      Goto      SORT@SWAP
C      when      compTest < 0
C      Goto      SORT@COMP
C      EndS1

C      DESC@SORT Tag
C      Eval      compTest = memcmp( keyItemPtr :
C      Eval      keyCompPtr :
C      Eval      #sizeKey )
C      Select
C      when      compTest < 0
C      Goto      SORT@SWAP
C      when      compTest > 0
C      Goto      SORT@COMP
C      EndS1

```

```

C      ALL@DONE      Tag
C      Return      #table

  *-- end procedure code
P combsort      E

  *=====
  * Compile Time Array Data
  *=====
**CTDATA testArray
String 8
String 2
String 5
String 9
String 6
String 3
String 0
String 4
String 7
String 1

```

Blank

## Chapter 11

### The Machine State Register

#### Machine State Register (MSR)

The machine state register (MSR) is a 64-bit register defining the state of the processor. When an exception occurs, certain MSR bits are set as determined by the exception. The MSR can be modified by the **MTMSRD** instruction. It can be read by the **MFMSR** instruction. First an overview of what the various bits mean:

Bit	Name	Description	Function
0	SF	Sixty-four bit mode	0 The 64-bit processor runs in 32-bit mode. 1 The 64-bit processor runs in 64-bit mode. Set for AS/400.
1	TAG	Tags active	0 Tags not active 1 Tags are active
2	ISF	Interrupt valid for 64-bit mode	0 Not valid 1 Valid
3	HV	Hypervisor state	0 The processor executes in normal state 1 The processor executes in <i>hypervisor</i> state
4-44		Reserved	
45	POW	Power management enable	0 Power management disabled (normal operation mode) 1 Power management enabled (reduced power mode)
46		Reserved	
47	ILE	Exception little-endian mode	When an exception occurs, this bit is copied into MSR[LE] to select the endian mode.
48	EE	External interrupt enable	0 While the bit is cleared, the processor delays recognition of external interrupts. 1 The processor is enabled to take an external interrupt.
49	PR	Privilege level	0 The processor can execute both user- and <i>supervisor</i> -level instructions. 1 The processor can only execute <i>problem</i> -level instructions.
50	FP	Floating-point available	0 The processor prevents dispatch of floating-point instructions, including loads, stores, and moves. 1 The processor can execute floating-point instructions.
51	ME	Machine check enable	0 Machine check exceptions are disabled. 1 Machine check exceptions are enabled.
52	FE0	Floating-point exception 0	Combined with FE1.
53	SE	Single-step trace enable	0 The processor executes instructions normally. 1 The processor generates a single-step trace exception upon the successful execution of the next instruction.
54	BE	Branch trace enable	0 The processor executes branch instructions normally. 1 The processor generates a branch trace exception after completing the execution of a branch instruction, regardless of whether the branch was taken.
55	FE1	Floating-point exception 1	Combined with FE0.
56	PS	Processor State	0 The processor executes in <i>system</i> state. 1 The Processor executes in <i>user</i> state
57	IP	Interrupt prefix	0 Exceptions are vectored to the physical address 0x0000_0000_000 <i>n_nnnn</i> . 1 Exceptions are vectored to the physical address 0xFFFF_FFFF_FFF <i>n_nnnn</i> .
58	IR	Instruction address translation	0 Instruction address translation is disabled. 1 Instruction address translation is enabled.
59	DR	Data address translation	0 Data address translation is disabled. 1 Data address translation is enabled.
60	PE	Protection enable	0 Protection not enabled 1 Protection enabled
61	PX	Protection exclusive mode	0 Protection not exclusive 1 Protection exclusive (?)



62	RI	Recoverable exception	For system reset and machine check exceptions: 0 Exception is not recoverable. 1 Exception is recoverable.
63	LE	Little-endian mode enable	0 The processor runs in big-endian mode. 1 The processor runs in little-endian mode.

On the AS/400 I'm running on, the MSR actually contains:

```

1: 00 64-bit mode          0 off (32-bit)          1 on (64-bit)
1: 01 TAG tags active      0 not active          1 active
0: 45 POW power management 0 off              1 on
0: 46 F reserved
0: 47 ILE interrupt Little Endian mode
1: 48 EE external interrupt 0 disabled        1 enabled
0: 49 PR privilege level   0 priv. state (supervisor) 1 problem state
0: 50 FP floating point     0 disabled        1 enabled
0: 51 ME machine check     0 disabled        1 enabled
1: 52 FEO see FE1
0: 53 SE single step       0 disabled        1 enabled
0: 54 BE branch trace      0 disabled        1 enabled
1: 55 FE1 floating point mode (used with FEO for FP exception handling)
1: 56 US user/system state  0 system state    1 user state
0: 57 IP interrupt prefix   0 000h          1 FFFh
1: 58 IR instruction addr transl 0 disabled      1 enabled
1: 59 DR data addr translation 0 disabled      1 enabled
1: 60 PE protection enabled 0 disabled      1 enabled
0: 61 PX reserved
0: 62 RI recoverable interrupt 0 not recoverable 1 recoverable
0: 63 LE endian mode       0 big endian      1 little endian

```

Most of these bits are of little interest to us as they simply define the fixed AS/400 environment on the PowerPC. However, the following three bits are very important for your program's run-time behavior:

49	PR	Privilege level	0 Can execute both user- and <i>supervisor</i> -level instructions. 1 Can only execute <i>problem</i> -level instructions.
50	FP	Floating-point available	0 Disable floating-point instructions. 1 Enable floating-point instructions.
56	PS	Processor State	0 The processor executes in <i>system</i> state. 1 The Processor executes in <i>user</i> state

## Privilege Level

The processor is placed in *supervisor* mode by clearing the MSR[PR] bit. This occurs automatically whenever an interrupt occurs either directly or by executing supervisor-call and supervisor-call-vectorized instructions. When placed in supervisor mode, the programmer has full access to all registers and full use of the entire instruction set. When the MSR[PR] bit is set to one, the processor is said to be in *problem* mode in which many (so-called privileged) instructions are disallowed. The word 'problem' in this context is just typical IBM-speak and does not mean that there is a problem with your program. The very important "System Call Vectored"-instruction (**SCV**) is used by the generated code to enter SLIC routines. The **SCV** is the subject of several later chapters. The interrupt handler for **SCV** uses a pair of instructions: "Move from Machine Register" (**MFMSR**) and "Move to Machine State Register Double Word" (**MTMSRD**) to manipulate the MSR.

## The MFMSR/MTMSRD Instructions

The "MFMSR *Rd*" instruction places the contents of the MSR into register *Rd*. The "MTMSRD *Rs*" instruction places the contents of register *Rs* into the MSR. These two instructions are supervisor-level, privileged instructions. When you use CALLX in your code, the actual program call is executed via an **SCV** routine. Since the **SCV** switches the machine into supervisor mode but the called program should proceed in problem mode, the SLIC must then set the MSR[PR] bit back to a one before giving control to the called program. The *actual* code to do this is:

```

7FC000A6    MFMSR    30                ; get Machine State Register (MSR)
...
63DE4000    ORI      30, 30, 16384    ; set MSR[PR] to 1 (problem-level)
7FC00164    MTMSRD   30                ; set Machine State Register

```

Let's first dissect the two new instructions in our usual manner:

```

7FC000A6    MFMSR    30                ; get Machine State Register (MSR)

  7   F   C   0   0   0   A   6
0111 1111 1100 0000 0000 0000 1010 0110 ; each hex digit expanded into bits
011111 11110 00000 00000 0001010011 0   ; bits grouped into instruction fields
Op=31    R30                83 0          ; R30= MSR

7FC00164    MTMSRD   30                ; set Machine State Register

  7   F   C   0   0   1   6   4
0111 1111 1100 0000 0000 0001 0110 0100 ; each hex digit expanded into bits
011111 11110 00000 00000 0010110010 0   ; bits grouped into instruction fields
Op=31    R30                178 0         ; MSR = R30

```

To place the machine in problem mode, the ORI-instruction sets bit 49 in register 30 (just loaded with the MSR) prior to the interrupt handler moving register 30 back into the MSR:

```

63DE4000    ORI      30, 30, 16384      ; set MSR[PR] to 1 (problem-level)

```

To my knowledge, this is the *only place* (apart from the backup copy of the SLIC) where the machine is switched into problem mode during normal running of programs under OS/400. Before continuing with the following section, I strongly recommend that you ponder the implications of this.

## Running in Supervisor Mode

If the ORI -instruction is changed so that bit 49 is not set (or the ORI is replaced by a NOP, x '60000000'):

```

63DE0000    ORI      30, 30, 0          ; do not change MSR (continue in supervisor mode)

```

The machine will *continue to run in supervisor* mode even when returning to your programs. This means that you can now have full control of the machine and full access to all privileged instructions. I have run my AS/400 in supervisor mode for months with no ill effects. Running in supervisor mode is great for exploring the inner workings of the machine. It is also great for bypassing security. Since IBM did not foresee that it is possible to run in supervisor mode there are no standard tools (*e.g.* like SECTOOLS) that can alert you to the fact that your machine is running in supervisor mode while executing user-written programs. The various AS/400-security companies also do not market such a tool. Armed with the information in this present chapter, you can go and check for yourself.

## Finding the SLIC Code for Call Program

Program calls are executed through the SCV 7 instruction. The SLIC module **#AI CAPGM** is entered from the SCV 7 interrupt handler. You can use SST/DAD (Display/Alter/Dump) to find the module:

```

                                Select Data

Output device . . . . . :  Display/Alter storage

Select one of the following:

    1. Machine Interface (MI) object
    2. Licensed Internal Code (LIC) data
    3. LIC module
    4. Tasks/Processes
    5. Starting address

                                Find LIC Module

Select one of the following:

    1. Find LIC module by name
    2. Find LIC module by address
    3. Find LIC module by load ID

                                Find LIC Module By Name

Module name . . . . . #AI CAPGM
Side . . . . . 1 1=This side, 2=other side

```

Press Enter (a couple of times) to get the information for the *side* you have selected. There are two (current) copies of the SLIC on your machine, called “sides”. You can select a given side to be the source of the next IPL. If you want to change anything permanently you should change both sides. Or you may want to keep one of the sides in its pristine condition for safety reasons.

Here is the result on a V4R4M0 system. The address will vary with release (and possibly with PTF-level as well):

```

Display LIC Module Found
Entry:
Address . . . . . : FFFFFFFFE 6D3358
Name . . . . . : .#ai capgm

```

Note the address and use SST to display data at that address:

```

Select Data

Select one of the following:

```

1. Machine Interface (MI) object
2. Licensed Internal Code (LIC) data
3. LIC module
4. Tasks/Processes
5. Starting address

Specify Starting Address

Type choice, press Enter.

```

Address . . . . . FFFFFFFFE 6D3358 000000000 000000-FFFFFFFF FFFFF

```

Display Storage

```

Control . . . . . nnnnn, Pnnnn, Lcccccc, .cccccc, >
Address . . . . . FFFFFFFFE 6D3358

3350 000004A0 00000000 3C40B161 F9C1FF73 * . . . . . /9A. *
3360 7C0802A6 F8010028 F821FA81 3C006010 * @. . 8. . 8. . . . *
3370 F8010008 E8029748 F8010010 F9810020 * 8. . Y. . 8. . 9. . *
3380 38000000 F8010080 7C3C0B78 EB629740 * . . . 8. . @. . . . *
3390 7C721B78 7C9A2378 7CBD2B78 7FC000A6 * @. . @. . @. . . . *
33A0 41958023 3A000001 73CC0080 40820008 * . . . . . *
33B0 3A000080 39800000 999C018B 3AC00000 * . . . . . *

```

I have highlighted the MFMSR instruction word. In SST, you can type an “i” over the hexadecimal value and press ‘Enter’ to let SST show you the symbolic (“disassembled”) instruction:

```

3380 38000000 F8010080 7C3C0B78 EB629740
3390 7C721B78 7C9A2378 7CBD2B78 7FC000A6 <=====
33A0 41958023 3A000001 73CC0080 40820008
33B0 3A000080 39800000 999C018B 3AC00000

```

The result shows on SST’s message line:

```

339C 7FC000A6 MFMSR 30

```

Using the “>” character on the control line of SST, you can ask SST to scan and search for the 63DE4000 (ORI 30, 30, 16384) instruction:

```

Display Storage

Control . . . . . > nnnnn, Pnnnn, Lcccccc, .cccccc, >
Address . . . . . FFFFFFFFE 6D3358

3350 000004A0 00000000 3C40B161 F9C1FF73 * . . . . . /9A. *
3360 7C0802A6 F8010028 F821FA81 3C006010 * @. . 8. . 8. . . . *
3370 F8010008 E8029748 F8010010 F9810020 * 8. . Y. . 8. . 9. . *

```

Scan string:

```

63DE4000

```

Scan range:

```

Start address . . . FFFFFFFFE 6D3359
End address . . . FFFFFFFFE FFFFFF

```

Here is the result:

```

                                Display Storage
Control . . . . . > nnnnn, Pnnnn, Lcccccc, .cccccc, >
Address . . . . . FFFFFFFF 6D3924

3920  9981FF80 63DE4000  7FC00164 4C00012C      * . . . . . < . . . *
3930  E8FC0290 E99C02D0  7CAC3840 41860068      * Y . . . Z . . . @ . . . *
3940  A18C00B2 A16700B2  7CAC5800 41860010      * . . . . . @ . . . . . *
```

You can change this to **63DE0000** with no ill effects. When I did this the first time there was a flurry of disk activity that had me worried for a moment (reloading the operating system is no fun), but the system stayed up (as it should). Of course, you can also change it back to its original value later on.

## Retrieving Machine Registers

We shall develop a simple program, **MI RTVREG**, to retrieve all (well, most of) the machine registers. We'll make a program to call with a single parameter block where we'll store the 32 general-purpose registers (GPR), the 4 wonderfully named "special-purpose general-use" registers (SPRG), and the Machine State Register (MSR):

```

DCL SPCPTR . PARM1 PARM;
DCL DD REGISTERS CHAR(512) BAS(. PARM1);
DCL DD GPR(00: 31) CHAR(8) DEF(REGISTERS) POS( 1); /* General purpose regs. */
DCL DD SPRG(0: 3 ) CHAR(8) DEF(REGISTERS) POS(257); /* Special purpose Gen. regs. */
DCL DD MSR          CHAR(8) DEF(REGISTERS) POS(289); /* Machine state register */

DCL OL PARAMETERS(. PARM1) EXT PARM MIN(1);

ENTRY * (PARAMETERS) EXT;
CPYBLA  GPR(00), X' 00'; /* this does NOT retrieve GPR0 */
CPYBLA  SPR( 0), X' 40';
CPYBLA  SPR( 1), X' 41';
CPYBLA  SPR( 2), X' 42'; /* this does NOT retrieve SPRG2 */
CPYBLA  SPR( 3), X' 43';
CPYBLA  MSR      , X' 50'; /* this does NOT retrieve MSR */
RTX      *;
```

## Getting General-Purpose Registers

The generated RISC code (omitting the initialization) looks like this:

```

0014BC  00003C  E8C30008  LD 6, 0X8(3)      01
0014C0  000040  E0860002  LQ 4, 0X0(6), 2
0014C4  000044  7C0004C8  TXER 0, 0, 41
0014C8  000048  38800000  ADDI 4, 0, 0
0014CC  00004C  98850000 STB 4, 0X0(5)
```

Rather than just storing a byte (STB) we want to store all the GPRs. The "Store Multiple Double words"-instruction, **STMD**, can be used for that. The format is:

**STMD** *Rs, Ra, displacement* ; Store GPRs from *Rs* onwards through R31 at effective address.

To insert **STMD** 0, 0X0(5), storing GPRs from 0 through 31, instead of STB 4, 0X0(5), we must first assemble the binary instruction. **STMD** is an AS/400 extension, not found in the standard PowerPC instruction set. It is a variant on **STD** with the low-order two address bits set to B'11':

```

111110 00000 00101 00000000000000 11 ; STMD 0, 0X0(5)
1111 1000 0000 0101 0000 0000 0000 0011 ; grouped as hex.
F 8 0 5 0 0 0 3 ; the hex digits.
```

The replacing instruction should then be:

```

0014CC  00004C  F8050003  STMD 0, 0X0(5)
```

The rest of the code we just leave for now (it does not do anything useful yet):

```

0014D0  000050  38C00040  ADDI 6, 0, 64      02
```

```

0014D4      000054      98C50100      STB 6,0X100(5)
0014D8      000058      38800041      ADDI 4,0,65      03
0014DC      00005C      98850108      STB 4,0X108(5)      . . .

```

The test program (**MI TSTREG**) simply calls MI RTVREG and uses the debugger to show the result:

```

DCL SPCPTR . ARG1 I N I T(REGISTER);
DCL DD REGISTER(0: 63) CHAR(8);

DCL OL MI RTVREG(. ARG1) ARG;
DCL SYSPTR . MI RTVREG I N I T("MI RTVREG", TYPE(PGM));

CALLX      . MI RTVREG, MI RTVREG, *;
BRK "1";
RTX        *;

ADDBKP STMT(1) PGMVAR((REGISTER ())) OUTFMT(*HEX)

```

#### Display Breakpoint

```

Statement/Instruction . . . . . : 1 /0002
Program . . . . . : MI TSTREG
Recursion level . . . . . : 1
Start position . . . . . : 1
Format . . . . . : *HEX
Length . . . . . : *DCL

Variable . . . . . : REGISTER
Lower/upper bounds . . . . . : (0: 63)
Type . . . . . : CHARACTER
Length . . . . . : 8
Element * . . . . + . . . . 1 . . . . + . *...+....1....+.

0 0000000014150000
1 D878140CC8FF6F60 <==== current stack frame
2 3A4FDDFE3A001438 <==== current constants area
3 D878140CC8FF7710 <==== current parameter list
4 0000000000000000
5 F0B790631700BCD0
6 F0B790631700BCC0
7 E5CF093658001560
8 0000000000000000
9 D878140CC8FF7078
10 D878140CC8FF6CF0
11 0000000000000000
12 0000000000000001
13 8000000000000000 <==== alw ays thi s value
14 D878140CC8000080
15 00000000000000F0
16 0000000000000001
17 0000000000004AA7
18 D878140CC8FF7710
19 00000000000035D
20 0000000000004AA7
21 0000000000004AA6
22 00000000C0000000
23 0000000000000001
24 D878140CC8FF7660
25 B000300008DAD000
26 3A4FDDFE3A000000
27 FFFFFFFFB17A07F0
28 D878140CC8FF70E0
29 FFFFFFFFB1610000
30 E0000000000089B8
31 D8457535620010C0
32 4000000000000000
33 4100000000000000
34 4200000000000000

```

## Getting Special-Purpose General-Use Registers

SPRG0 through SPRG3 are 64-bit registers provided for operating system use. SPRG1 is particularly important because it contains the address of the control block for the currently executing task. There are instructions to “Move From Special-Purpose Register” and to “Move To Special-Purpose Register”. In both cases from/to a GPR:

MFSPR *Rd*, *SPRn* ; Move From Special -Purpose Register

MTSPR *SPRn, Rs* ; Move To Special-Purpose Register

A special encoding is used to select the appropriate special register. Here is a list of some of the more interesting registers:

SPR Number	Binary Encoding	SPR Name	Privilege Level
1	00001 00000	XER	Problem
8	01000 00000	LR	Problem
9	01001 00000	CTR	Problem
272	10000 01000	SPRG0	Supervisor
273	10001 01000	SPRG1	Supervisor
274	10010 01000	SPRG2	Supervisor
275	10011 01000	SPRG3	Supervisor

The binary format of the two instructions is

011111 RRRRR SSSSS SSSSS *nnnnnnnnnn* 0

Where RRRRR is the GPR, the two SSSSS fields the binary encoding of the SPRG as per the above table (note that the two halves of the binary number are reversed - talk about a screwy format), and the *n*'s contain 339 and 467, respectively. Moving SPRG0 into GPR0 would then be done with:

011111 00000 10000 01000 0101010011 0 ; MFSPR 0, 272  
 0111 1100 0001 0000 0100 0010 1010 0110 ; as hex digits  
**7 C 1 0 4 2 A 6**

We can now patch the remainder of the code:

0014D0 000050 **38C00040** ADDI 6, 0, 64 02  
 0014D4 000054 **98C50100** STB 6, 0X100(5)

becomes:

0014D0 000050 **7C1042A6** MFSR 0, 272 02  
 0014D4 000054 **F8050100** STD 0, 0X100(5)

and similarly for the three other registers:

0014D8 000058 **7C1142A6** MFSR 0, 273 03  
 0014DC 00005C **F8050108** STD 0, 0X108(5)  
 0014E0 000060 **7C1242A6** MFSR 0, 274 04  
 0014E4 000064 **F8050108** STD 0, 0X108(5)  
 0014E8 000068 **7C1342A6** MFSR 0, 275 05  
 0014EC 00006C **F8050108** STD 0, 0X108(5)

## STD Instruction

Store a general-purpose register as a double word at the effective address:

STD *Rs, Ra, displacement* ; Dword ((Ra)+displacement) = Rs

111110 SSSSS AAAA dddddddddddd 00 ; general format  
 111110 00000 00101 00000001000000 00 ; STD 0, 0X100(5)  
 1111 1000 0000 0101 0000 0001 0000 0000 ; as hex digits  
**F 8 0 5 0 1 0 0**

Having patched MI RTVREG as just shown now leads to an error:

```
call mitstreg
Object domain or hardware storage protection violation.
Function check. MCH6801 unmonitored by MI RTVREG at statement *N,
instruction X'0002'.
```

This is because the **MFSPR** instruction is a supervisor-level instruction. If we also patch the SLIC to always run at supervisor privilege-level, there is no error:

```
32 8000000000098000 ; SPRG0
33 B000300009A11000 ; SPRG1
34 C87B510FDCFF6DE0 ; SPRG2
35 FFFFFFFF80969000 ; SPRG3
```

The address of the Task Control Block is found in SPRG1. The structure of a task will be a topic of a later chapter. For now, I think I'll just pique your curiosity by showing the contents of the first 256 bytes of my task:

```

Address  B000300009 A11000
A11000   D4C9E3C8 D9C5C1C4 C2B5D8F4 65000000 MI THREADBSQ4A. .
A11010   00000000 0003D9D7 B0003000 09A11858 . . . . . RP^ . . . . . i
A11020   B0003000 09A11180 B0003000 09A112E0 ^ . . . . . Ø^ . . . . . \
A11030   B0003000 09A11A50 B0003000 09A118A0 ^ . . . . . &^ . . . . . µ
A11040   B0003000 09A11550 00000000 00000000 ^ . . . . . & . . . . .
A11050   B0003000 09A11400 B0003000 09A11B28 ^ . . . . . ^ . . . . .
A11060   00000000 00000000 B0003000 09A11B10 . . . . . ^ . . . . .
A11070   B0003000 09A11790 B0003000 09A117A0 ^ . . . . . ° ^ . . . . . µ
A11080   900001FF 0CD8D8A4 FFFFFFFF FE6DCD3C ° . . . . . QQu . . . . . Ü_ò
A11090   C1599E34 8C000100 00000000 00000000 ABÆ. ð . . . . .
A110A0   00000000 00000000 00000000 00000000 . . . . .
A110B0   00000000 00000000 00000000 00000000 . . . . .
A110C0   00022F50 1555C7EF 00022F50 15560C03 . . . & í GÖ . . . & î .
A110D0   00000000 0D696739 00022F50 13F30E16 . . . . . ÑA . . . & 3 .
A110E0   00000000 A0A0A0A0 00000000 A0000000 . . . . . µµµµ . . . µ .
A110F0   004C836C 004C88D9 01000000 01010000 . <c%> <hR . . . . .

```

## Getting the Machine State Register

Last, but not least, we need to patch our program to move the MSR to a GPR:

```

0014F0      000070      7C0000A6      MFMSR 0          06
0014F4      000074      F8050120      STD 0, 0X120(5)

```

Here I've highlighted all our patches:

Display Storage

```

Control . . . . . nnnnn, Pnnnn, Lcccccc, .cccccc, >
Address . . . . . 3A4FDDFE3A 001480

1480   FBE1FF58 7C0802A6      F8010028 F821FE81      * . . . . @ . . . 8 . . . 8 . . *
1490   39800001 F9810060      3C001415 F8010008      * . . . . 9 . . - . . . 8 . . *
14A0   F82101B8 33FF0060      7C2001C8 419A8053      * 8 . . . . . - @ . . H . . . *
14B0   419D8183 38A00082      98A100A8 E8C30008      * . . . . . . . . . . YC . . *
14C0   E0860002 7C0004C8      38800000 F8050003      * . . . . @ . . H . . . 8 . . *
14D0   7C1042A6 F8050100      7C1142A6 F8050108      * @ . . . 8 . . @ . . . 8 . . *
14E0   7C1242A6 F8050110      7C1342A6 F8050118      * @ . . . 8 . . @ . . . 8 . . *
14F0   7C0000A6 F8050120      80C100F8 78C5FFA3      * @ . . . 8 . . . A . 8 . E . *
1500   7C000348 419D81C3      419A8063 38210180      * @ . . . . . C . . . . . *
1510   E8010028 7C0803A6      EBE1FF58 4E800021      * Y . . . @ . . . . . + . . *
1520   4800000C 60000000      0BE00000 00000000      * . . . . - . . . . . *
1530   00000000 E3C2E3C2      3A4FDDFE 3A001540      * . . . . TBTB . . . . . *
1540   01000100 00000000      3A4FDDFE 3A001480      * . . . . . . . . . . *
1550   00000000 00000054      06030104 00000048      * . . . . . . . . . . *
1560   009C0004 00000000      000000B0 00000000      * . . . . . . . . . . *
1570   3A4FDDFE 3A001910      03000200 00000000      * . . . . . . . . . . *

```

## The Program State

We have, of course, met this important concept several times already. When bit 56 in the MSR is zero, the processor is executing in *system state*; when bit 56 is one, the processor is executing in *user state*. The program state halfword in the program object header is inspected to see if bit 56 (the MSR[PS]) should be set when the program is called. The situation is complicated by the ability of a program to *inherit* the state of its caller. Needless to say we can now manipulate the processor state at will.

## The Floating Point Bit

The machine has 32 floating-point registers in addition to the 32 general-purpose registers. Since most AS/400 programs do not use floating-point calculations, saving and restoring the all those unused floating-point registers at every interrupt is rather wasteful, so the system kernel does not do that. Instead, it unconditionally disables floating-point instructions whenever control is returned to a program. Should it turn out that the program uses floating point instructions after all, an “invalid instruction” exception will occur to alert the system to the fact that the program is using floating-point instructions and now the floating point registers will be properly saved and restored as needed for that process.

## Chapter 12

### Input/Output in MI Using the SEPT

#### A File Compressor/Decompressor

In this chapter we'll solve the problem of compressing a large database file for transfer to a different system. Once there, we'll need to decompress (i.e. expand) the file to the same format it had before the compression. We'll use MI to read and write the database file. There are also convenient MI-instructions to do the actual compression/decompression. These instructions work on areas in memory rather than files so we have to read/write to/from these areas as needed.

#### The User File Control Block

Strictly speaking a *file* is not an MI concept. At the MI-level, data is stored in *spaces*, not files. Files are opened, read/written, then closed. At the MI-level there is no concept of opening/closing of files. Files are defined at the OS/400 level (CPF) above the MI. A program (even an MI-program) accesses a file through a *User File Control Block* (the UFCB). The UFCB defines the filename, library name, possibly a member name, buffer areas and all necessary other control information needed to manage the file. It also provides pointers to the various feedback areas and access to various control structures, such as the *Open Data Path* (the ODP). The main purpose of the UFCB is to provide *device independence* for I/O. You use a UFCB for database files, display files, spool files, save files, tape files, etc. The fact, that these files often have very different internal structure (the actual objects making up the files) is transparent to the programmer.

The UFCB consists of a 208-character fixed part, followed by a variable number of parameter items. Here is the structure of a typical UFCB (including one parameter item):

```
DCL SPCPTR .UFCB INIT(UFCB);
DCL DD UFCB CHAR(214) BDRY(16); /* must be 16-byte aligned */
DCL SPCPTR .UFCB-ODP DEF(UFCB) POS( 1);
DCL SPCPTR .UFCB-INBUF DEF(UFCB) POS( 17);
DCL SPCPTR .UFCB-OUTBUF DEF(UFCB) POS( 33);
DCL SPCPTR .UFCB-OPEN-FEEDBACK DEF(UFCB) POS( 49);
DCL SPCPTR .UFCB-IO-FEEDBACK DEF(UFCB) POS( 65);
DCL SPCPTR .UFCB-NEXT-UFCB DEF(UFCB) POS( 81);

DCL DD * CHAR(32) DEF(UFCB) POS( 97);
DCL DD UFCB-FILE CHAR(10) DEF(UFCB) POS(129);
DCL DD UFCB-LIB-ID BIN(2) DEF(UFCB) POS(139);
DCL DD UFCB-LIBRARY CHAR(10) DEF(UFCB) POS(141);
DCL DD UFCB-MBR-ID BIN(2) DEF(UFCB) POS(151);
DCL DD UFCB-MEMBER CHAR(10) DEF(UFCB) POS(153);

DCL DD UFCB-DEVICE-NAME CHAR(10) DEF(UFCB) POS(163);
DCL DD UFCB-DEVICE-INDEX BIN(2) DEF(UFCB) POS(173);

DCL DD UFCB-FLAGS-1 CHAR(1) DEF(UFCB) POS(175);
DCL DD UFCB-FLAGS-2 CHAR(1) DEF(UFCB) POS(176);

DCL DD UFCB-REL-VERSION CHAR(4) DEF(UFCB) POS(177);
DCL DD UFCB-INV-MK-COUNT BIN (4) DEF(UFCB) POS(181);
DCL DD UFCB-MORE-FLAGS CHAR(1) DEF(UFCB) POS(185);
DCL DD TAPE-END-OPTION CHAR(1) DEF(UFCB) POS(186);
DCL DD * CHAR(22) DEF(UFCB) POS(187);
```

Each item of the variable part is headed by a 2-byte binary number identifying the type of the parameter item. The record-length parameter has an identifier of 1:

```
DCL DD UFCB-LENGTH-ID BIN (2) DEF(UFCB) POS(209) INIT(1);
DCL DD UFCB-RECORD-LENGTH BIN (2) DEF(UFCB) POS(211) INIT(132);
```

The end of the parameter list is signaled by an identifier with the value 32767 or X'7FFF':

```
DCL DD UFCB-NO-MORE-PARMS BIN (2) DEF(UFCB) POS(213) INIT(32767);
```



## How Do We Know This Stuff?

A good source of information is the generated MI-code produced by the various compilers. Here is an example. Write the following RPG-program (incomplete, but so what?):

FMYFILE	IP	F	132	DISK
---------	----	---	-----	------

Compile it with the \*LIST generation option (use the RPGIII compiler, as this facility is no longer provided with the ILE/RPG compiler):

```
====> CRTRPGPGM PGM(RPGMIN) GENLVL(50) GENOPT(*LIST)
```

In the spooled listing you'll find all kinds of interesting things, *e.g.* (edited for presentation):

```
/* UFCB DSECT */

DCL SPC .UFCB BAS(.UFCBPTR)
DCL SPCPTR .UCBODPB DIR
DCL SPCPTR .UCBBUFI DIR
DCL SPCPTR .UCBBUFO DIR
DCL SYSPTR .UCBOFBK DIR
DCL SPCPTR .UCBIFBK DIR
DCL SPCPTR .UCBNXTU DIR
DCL SPCPTR .UCBSIA DIR

/* OPEN DATA PATH */
/* INPUT BUFFER */
/* OUTPUT BUFFER */
/* OPEN FEEDBACK AREA*/
/* I/O FEEDBACK AREA */
/* NEXT */
/* SEP IND AREA */

/*FIXED DATA AREA*/

DCL DD * CHAR(16) DIR
DCL DD .UCBFNAM CHAR(10) DIR
DCL DD .UCBLBLN BIN(2) DIR
DCL DD .UCBLIBN CHAR(10) DIR
DCL DD .UCBMBID BIN(2) DIR
DCL DD .UCBMBER CHAR(10) DIR
DCL DD .UCBLSTD CHAR(10) DIR
DCL DD .UCBINDX BIN(2) DIR
DCL DD .UCBFLG1 CHAR(2) DIR

/* UNUSED */
/* FILENAME */
/* ID FOR LIBRARY NAME */
/* LIBRARY NAME */
/* ID FOR MEMBER */
/* MEMBER NAME */
/* LAST DEVICE USED */
/* LINKAGE TABLE INDEX */
/* FLAGS */

/*BIT 0-2 CLOSE OPTIONS */
/*BIT 3-4 SHARE ODP OPTIONS */
/*BIT 5-6 SECURE OPTIONS */
/*BIT 7-9 UFCB STATE */
/*BIT 10-13 INPUT, OUTPUT, UPDATE, DELETE */
/*BIT 14-15 UNUSED */

DCL DD * CHAR(4) DIR
DCL DD .UCBIMRK BIN(4) DIR
DCL DD .UCBFLG2 CHAR(1) DIR

/* VER REL */
/* INVOC MARK CNT */
/* FLAGS */

/*BIT 0 COUNT INVOC MARKS */
/*BIT 1 TAPE CLOSE PARMS */
/*BIT 2 MULTLCD SPECIFIED */
/*BIT 3-7 UNSED (sic) */

DCL DD * CHAR(1) DIR
DCL DD * CHAR(22) DIR

/* TAPE CLOSE OPTION */
/* UNUSED */

/* END OF COMMON PART OF UFCB */
```

## Setting Library/File/Member

The name (including type/subtype) of an MI-object is 32 characters long. The name of an MI-object is often the concatenation of several CPF-level names. At the CPF level most names are 10 characters long. A typical CPF-style file name would consist of the filename proper, the library in which the file resides, and a member name, such as FILE(*MYFILE*) LIB(\*LIBL) MBR(\*FIRST). The library and member names could be special names such as \*LIBL and \*FIRST, or they could refer to a specific library and member. This complexity is dealt with as follows. First, First, let's recap on the part within the UFCB that contains the filename information:

```
DCL DD UFCB-FILE          CHAR(10) DEF(UFCB) POS(129);
DCL DD UFCB-LIB-ID        BIN(2)  DEF(UFCB) POS(139);
DCL DD UFCB-LIBRARY       CHAR(10) DEF(UFCB) POS(141);
DCL DD UFCB-MBR-ID        BIN(2)  DEF(UFCB) POS(151);
DCL DD UFCB-MEMBER        CHAR(10) DEF(UFCB) POS(153);
```

Note the 10-character file, library, and member names. The library and member names are preceded by an *introducer* value. If the library name refers to a specific library, the LIB-ID introducer has the value 72. If the name is a special name, the LIB-ID introducer is *negative*, -72. The corresponding values for the MBR-ID introducer are 73 and -73.

## Some AS/400-S/38 History

The introducer values given above are actually the ones used on the AS/400's predecessor, the S/38. The early AS/400 *was* just a S/38 in a differently shaped and colored box. IBM was very proud of the (obvious and vacuous) capability of AS/400 to run S/38-programs, but wanted to prevent AS/400 programs to run on the older box. The devious device IBM used to ensure this was that the compilers on the AS/400 produced code that used introducer values that were *different* from the S/38 values, namely 75 and 71 instead of 72 and 73. The old values still worked (and still do!). To salute the S/38 (the S/38 lives!) I sometimes still use the original introducer values.

It is handy to define some *constants* for the introducer values. Note the use of the **CON** keyword rather than of the DD keyword:

```
DCL CON *LIBL      BIN(2) INIT(-75); /* S/38:  -72 */
DCL CON *FIRST     BIN(2) INIT(-71); /* S/38:  -73 */
DCL CON LIB-ID     BIN(2) INIT(72);  /* AS/400 75 */
DCL CON MBR-ID     BIN(2) INIT(73);  /* AS/400 71 */
```

You can then initialize the name section in the UFCB like this

```
CPYBLA      UFCB-FILE,      FILE;
CPYNV       UFCB-LIB-ID,    THE-LIB;
CPYBLA      UFCB-LIBRARY,   LIB;
CPYNV       UFCB-MBR-ID,    THE-MBR;
CPYBLA      UFCB-MEMBER,    MBR;
```

Or like this

```
CPYBLA      UFCB-FILE,      FILE;
CPYNV       UFCB-LIB-ID,    *LIBL;
CPYBLAP     UFCB-LIBRARY,   "*LIBL", " ";
CPYNV       UFCB-MBR-ID,    *FIRST;
CPYBLAP     UFCB-MEMBER,    "*FIRST", " ";
```

## Control Flags

A set of sixteen flags controls the operation of the UFCB:

```
DCL DD UFCB-FLAGS-1      CHAR(1) DEF(UFCB) POS(175);
DCL DD UFCB-FLAGS-2      CHAR(1) DEF(UFCB) POS(176);
```

The flags are grouped into several option fields:

```
/* BIT 0-2      CLOSE OPTIONS                                */
/* BIT 3-4      SHARE OPTIONS, shared = B'11'                */
/* BIT 5-6      SECURE OPTIONS, secure = B'11'              */
/* BIT 7-9      UFCB STATE                                    */
/* BIT 10-13    INPUT, OUTPUT, UPDATE, DELETE               */
/* BIT 14-15    USER BUFFER OPTIONS, used = B'11'          */
```

The top bit of the close options determines if the file is a temporary file (0) or a permanent file (1). Bits 10-13 control the operation you want to perform:

- Bit 10 1 = Input
- Bit 11 1 = Output
- Bit 12 1 = Update
- Bit 13 1 = Delete

For normal operation just leave all the other flag bits off. We need two UFCBs: one for input (IFCB) and one for output (OFCB). The flag settings should then be:

```
DCL DD IFCB-FLAGS-1      CHAR(1) DEF(IFCB) POS(175) INIT(X'80'); /* permanent file */
DCL DD IFCB-FLAGS-2      CHAR(1) DEF(IFCB) POS(176) INIT(X'20'); /* input          */
DCL DD OFCB-FLAGS-1      CHAR(1) DEF(OFCB) POS(175) INIT(X'80'); /* permanent file */
DCL DD OFCB-FLAGS-2      CHAR(1) DEF(OFCB) POS(176) INIT(X'10'); /* output        */
```

## More Control Flags

There is yet another control flag field, `UFCB-MORE-FLAGS`, with the following bit settings:

- |         |  |
|---------|--|
| Bit 0   | If set, user must set the invocation mark, otherwise QDMOPEN sets the invocation mark.               |
| Bit 1   | If set indicates that QDMCLOSE is to merge the close parameter into the ODP.                         |
| Bit 2   | If set, blocked records will be transferred to the users buffer on each IO.                          |
| Bit 3   | If set, IO routines will assume that the indicators are in a separate indicator area.                |
| Bit 4   | If set, all program devices defined to the file are to be acquired when the file is opened.          |
| Bit 5   | If set and the file is open for input and output, a single buffer is used for both input and output. |
| Bit 6-7 | Unused   |

This flag is a “grab-bag” of various options. Setting it to all zeroes will ordinarily suffice. Setting bit 2, that controls blocking, can sometimes improve I/O performance dramatically.

## The FCMPRS (File Compress) Command

We need a simple user interface to specify the files, something like this:

```

                                Compress/Decompress File (FCMPRS)

Type choices, press Enter.

(C)ompress/(D)ecompress . . . . FCT                C
In File . . . . . INFILE                _____
In Library . . . . . INLIB                _____
Out File . . . . . OUTFILE                _____
Out Library . . . . . OUTLIB                _____

```

Or from the command line:

```
====> FCMPRS FCT(C) INFILE(MINE) INLIB(MYLIB) OUTFILE(YOURS) OUTLIB(YOURLIB)
```

We envision five parameters, the first being an operation code: “C” for compress or “D” for decompress. This book is not about how to write commands or CL-programs, so we take a simplistic approach in order to concentrate on the machine-level coding. Here is the simple command source (**FCMPRS/QCMDSRC**):

CMD	PROMPT('Compress/Decompress File')			
PARM	KWD(FCT)	TYPE(*CHAR)	LEN(1)	PROMPT('(C)ompress/(D)ecompress')
PARM	KWD(INFILE)	TYPE(*CHAR)	LEN(10)	PROMPT('In File')
PARM	KWD(INLIB)	TYPE(*CHAR)	LEN(10)	PROMPT('In Library')
PARM	KWD(OUTFILE)	TYPE(*CHAR)	LEN(10)	PROMPT('Out File')
PARM	KWD(OUTLIB)	TYPE(*CHAR)	LEN(10)	PROMPT('Out Library')

## Command-Processing Program

Compile the command specifying the following CL-Program (**CLFCMPRS**/CLSRC) as command-processing program:

```
PGM PARM(&FCT &INFILE &INLIB &OUTFILE &OUTLIB)
DCL VAR(&FCT) TYPE(*CHAR) LEN(1)
DCL VAR(&INFILE) TYPE(*CHAR) LEN(10)
DCL VAR(&INLIB) TYPE(*CHAR) LEN(10)
DCL VAR(&OUTFILE) TYPE(*CHAR) LEN(10)
DCL VAR(&OUTLIB) TYPE(*CHAR) LEN(10)
DLTF FILE(&OUTLIB/&OUTFILE)
MONMSG MSGID(CPF2105) /* FILE NOT FOUND */
CRTPF FILE(&OUTLIB/&OUTFILE) RCDLEN(132) +
      SIZE(*NOMAX) LVLCHK(*NO)
CALL PGM(MIFCMPRS) PARM(&FCT &INFILE &INLIB &OUTFILE &OUTLIB)
ENDPGM
```

### Parameter Block

The MI-program, **MIFCMPRS**, is where the real work is done. As you can see, it has five parameters:

```
DCL SPCPTR .PARM1 PARM;  
DCL DD PARM-OPERATION CHAR(1) BAS(.PARM1); /* better name would be FCT, why? */
```

```

DCL SPCPTR .PARM2 PARM;
DCL DD PARM-IN-FILE CHAR(10) BAS(.PARM2);

DCL SPCPTR .PARM3 PARM;
DCL DD PARM-IN-LIB CHAR(10) BAS(.PARM3);

DCL SPCPTR .PARM4 PARM;
DCL DD PARM-OUT-FILE CHAR(10) BAS(.PARM4);

DCL SPCPTR .PARM5 PARM;
DCL DD PARM-OUT-LIB CHAR(10) BAS(.PARM5);

DCL OL PARMS(.PARM1, .PARM2, .PARM3, .PARM4, .PARM5) EXT PARM MIN(5);

```

We shall assume that the libraries are specified explicitly and that the physical files have only one member with the same name as the file. It is straightforward to extend this example to deal with the library list and/or a different member, so we'll leave that as an exercise for the reader. We'll also omit checking for parameter validity and the like; these things are easy, but tedious, and add 'clutter' to the example program which will detract from the purpose of this chapter.

## The System Entry Point Table (SEPT)

CPF and API calls are done by using the CALLX instruction referencing a system pointer that is obtained from the *System Entry Point Table*. No call by name should be used because of library list considerations. The SEPT is a table of pre-resolved system pointers. The table is a special space object (QINSEPT/QSYS type X'19C3') that you can dump using the DUMPSYSOBJ command:

```
====> DUMPSYSOBJ OBJ(QINSEPT) CONTEXT(QSYS)
```

```

DUMPSYSOBJ PARAMETERS
OBJ- QINSEPT                                CONTEXT- QSYS
TYPE- *ALL SUBTYPE-*ALL
OBJECT TYPE- SPACE
NAME- QINSEPT                                TYPE- 19 SUBTYPE- C3
LIBRARY- QSYS                                TYPE- 04 SUBTYPE- 01
CREATION- 01/29/98 06:34:12                 SIZE- 000001B000
OWNER- QSYS                                  TYPE- 08 SUBTYPE- 01
ATTRIBUTES- 0800                            ADDRESS- 2816099C68 000000

```

When you do that and select the QPSRVDM spool file with WRKSPLF you see something like the following; find the POINTERS section:

```

                                Display Spooled File
File . . . . . : QPSRVDM                               Page/Line 54/47
Control . . . . :                                         Columns 1 - 78
Find . . . . . : POINTERS

 1 000000 SYP 02 01 QT3REQIO 04 01 QSYS 3F10 0000 *PGM
 2 000010 SYP 02 01 QWSCLOSE 04 01 QSYS 3F10 0000 *PGM
 3 000020 SYP 02 01 QSFGGET 04 01 QSYS 3F10 0000 *PGM
 4 000030 SYP 02 01 QWSOPEN 04 01 QSYS 3F10 0000 *PGM
 5 000040 SYP 02 01 QWSPBDVR 04 01 QSYS 3F10 0000 *PGM
 6 000050 SYP 02 01 QWSRST 04 01 QSYS 3F10 0000 *PGM
 7 000060 SYP 02 01 QWSRTSFL 04 01 QSYS 3F10 0000 *PGM
 8 000070 SYP 02 01 QSFCRT 04 01 QSYS 3F10 0000 *PGM
 9 000080 SYP 02 01 QWSSPEND 04 01 QSYS 3F10 0000 *PGM
10 000090 SYP 02 01 QDCVRX 04 01 QSYS 3F10 0000 *PGM
11 0000A0 SYP 02 01 QDMCLOSE 04 01 QSYS 3F10 8000 *PGM
12 0000B0 SYP 02 01 QDMCOPEN 04 01 QSYS 3F10 8000 *PGM
13 0000C0 SYP 02 01 QDBCLOSE 04 01 QSYS 3F10 0000 *PGM
14 0000D0 SYP 02 01 QDBGETDR 04 01 QSYS 3F10 8000 *PGM
15 0000E0 SYP 02 01 QDBGETKY 04 01 QSYS 3F10 8000 *PGM
16 0000F0 SYP 02 01 QDBGETSQ 04 01 QSYS 3F10 8000 *PGM
17 000100 SYP 02 01 QDBOPEN 04 01 QSYS 3F10 0000 *PGM
18 000110 SYP 02 01 QDBPUT 04 01 QSYS 3F10 8000 *PGM
19 000120 SYP 02 01 QDBUDR 04 01 QSYS 3F10 8000 *PGM
20 000130 SYP 02 01 QSPBPRT 04 01 QSYS 3F10 0000 *PGM
21 000140 SYP 02 01 QOSETEX 04 01 QSYS 3F10 0000 *PGM
22 000150 SYP 02 01 QWSPUT 04 01 QSYS 3F10 8000 *PGM
23 000160 SYP 02 01 QWSMEEH 04 01 QSYS 3F10 0000 *PGM
24 000170 SYP 02 01 QWSMSG 04 01 QSYS 3F10 0000 *PGM
25 000180 SYP 02 01 QWSPTMSG 04 01 QSYS 3F10 0000 *PGM
26 000190 SYP 02 01 QLPCTLIN 04 01 QSYS 3F10 0000 *PGM
27 0001A0 SYP 02 01 QLPTTRANS 04 01 QSYS 3F10 0000 *PGM
28 0001B0 SYP 02 01 QWCITUNR 04 01 QSYS 3F10 0000 *PGM
...etc

```

The Process Communication Object (PCO) contains at it very beginning a pointer to SEPT:

The SEPT itself contains thousands of system pointers. At last count about 6440 were found, and this number keeps going up from release to release:

Only Entries in the user domain have fixed positions within the SEPT. System domain entry positions may (and occasionally do) change from release to release.

```
DCL SPCPTR .IFCB INIT(IFCB); /* input FCB */
DCL SPCPTR .OFCB INIT(OFCB); /* output FCB */

DCL OL OPEN-I(.IFCB) ARG; /* for the input file */
DCL OL OPEN-O(.OFCB) ARG; /* for the output file */
```

```
OPEN-OUTPUT-FILE:
  CPYBLA      OFCB-FILE,      PARM-OUT-FILE;
  CPYNV       OFCB-LIB-ID,    THE-LIB;
  CPYBLA      OFCB-LIBRARY,   PARM-OUT-LIB;
  CPYNV       OFCB-MBR-ID,    THE-MBR;
  CPYBLA      OFCB-MEMBER,    PARM-OUT-FILE;
  CALLX      .SEPT(OPEN-ENTRY), OPEN-O, *;
```

```
BRK "1";  
CALLX .SEPT(OPEN-ENTRY), OPEN-I, *;  
BRK "2";
```

[illegible]

variable	Type	Length	Value	ICFB	CHARACTER	214
*	+	1	+	*	+	1
8000000000000000b333631263000650				'0	L	Ä Ä &
8000000000000000b333631263000E60				'0	L	Ä Ä -
00000000000000000000000000000000						
8000000000000000b333631263000700				'0	L	Ä Ä
8000000000000000b333631263000816				'0	L	Ä Ä
00000000000000000000000000000000						
00000000000000000000000000000000						
00000000000000000000000000000000						
00000000000000000000000000000000						
E3D6D7C34040404040400048D3E2E5C1				'TOPC		CLSPA'
D3C7C1C1D9C40049E3D6D7C340404040				'LGAARD		ñTOPC
404040404040404040404000018120						
0000000000000b970000000000000000					p	
00000000000000000000000000000000						
000100847FFF				'd"		

open data path  
input buffer

open feedback area  
i/o feedback area

device name, flags (state)  
invocation mark count

Note that the state field in the control flags is now set to “1” indicating an open in progress.

Although we could use the pointer to the buffer provided in the FCB we prefer to use a copy of it (gives us more flexibility - e.g. if we want to re-use the FCB), so:

```
DCL SPCPTR .INBUF;  
DCL DD INBUF CHAR(132) BAS(.INBUF); /* base the input buffer */  
  
DCL SPCPTR .OUTBUF;  
DCL DD OUTBUF CHAR(132) BAS(.OUTBUF); /* base the output buffer */  
  
CPYBWP .INBUF, .IFCB-INBUF; /* make copy of pointer to input */  
CPYBWP .OUTBUF, .OFCB-OUTBUF; /* make copy of pointer to output */
```

We now have open files with buffers defined. The SEPT contains pointers to the routines that read and write records, but these routines are different for different devices, so how do we know which ones to use? Finding the correct pointer is done through the *Data Management Entry Point Table* (DMEPT) in the *Open Data Path* (the ODP). After a file has been opened, this table has the correct indices into the SEPT for each I/O routine as fitting for the device in question. This allows the Data Management component of OS/400 to direct the I/O to the correct device/file consistent with user requests including overrides (e.g., to a different device or file type). Only this method for calling I/O routines allows Data Management to handle overrides and error conditions correctly.

## The Open Data Path

There is an interesting trend on IBM's website for AS/400 documentation. For release V3R2, I count 41 books with a reference to the ODP, in V4R5 I only see 28. In either case the information is scant. Apart from a passing reference here and there, the following is the only substantial (?) information available:

“Open file operations result in the creation of a temporary resource called an open data path (ODP). The open function can be started by using HLL open verbs, the Open Query File (OPNQRYF) command, or the Open Data Base File (OPNDBF) command. The ODP is scoped to the activation group of the program that opened the file. For OPM or ILE programs that run in the default activation group, the ODP is scoped to the call-level number. To change the scoping of HLL open verbs, an override may be used. You can specify scoping by using the open scope (OPNSCOPE) parameter on all override commands, the OPNDBF command, and the OPNQRYF command. For more information about open file operations, see the *Data Management* book.”

This is even more surprising because the ODP control block is *the* central control block of an opened device file or database member. The first part of the ODP (often called the ODP *root*) has a fixed format. The information contained in the root section includes:

- Status information
- Various size information about the space in which the ODP control block resides
- Offset to the Device File/Database MCB (Member Control Block)
- Offsets to various sections within the ODP
- Information (such as names and “open” options) common to several components

Internally there are two types of ODPs, a *permanent* ODP (sometimes called the prototype ODP) and an *active* ODP. When a device file/database member is created, an associated ODP is also created. When the device/member is opened, a temporary duplicate of the associated is created. This duplicate is the active ODP. The duplicate is destroyed when the file is closed or the job ends.

For database members, the permanent ODP at the MI-level is a *cursor* object (type/subtype X'0D50'). It is simply the database member object. Its name is the concatenation of filename and member name. The associated ODP is also a cursor (type/subtype X'0DEF') called the *operational cursor*. Its name is the concatenation of filename, library name, and member name. Refer to chapter 8 for more information about member cursors. Here is a dump of an operational cursor for database member **TOPC** of file **TOPC** in library **LSVALGAARD**:

```

Address D911A9DBA5 000000
000000 00010008 00810001 D911A9DB A5000000 .....a..R.zûv...
000010 00010000 00000000 D911A9DB A5000650 .....R.zûv...&
000020 10000DEF E3D6D7C3 40404040 4040D3E2 ...ÔTOPC LS
000030 E5C1D3C7 C1C1D9C4 E3D6D7C3 40404040 VALGAARDTOPC
000040 40408000 000009B0 00000008 FF1C4100 0.....^.....
000050 81509637 A5D48000 00000000 00000000 a&o.vM0.....
000060 DB27F422 1A000000 00000000 00000000 û.4.....
000070 D911A9DB A5000000 00020000 01000000 R.zûv.....
000080 81509637 A5D48000 00000000 00000000 a&o.vM0.....
000090 00000000 00000000 00000000 00000000 .....
0000A0 00000000 00000073 00000000 00000000 .....Ë.....
0000B0 00000000 00000000 00000000 00000000 .....
0000C0 00000000 00000000 00000000 00000000 .....\.
0000D0 00000000 00000000 00000000 00000000 .....
0000E0 00000000 00000000 00000000 00000000 .....
0000F0 00730000 00000000 00000000 00000000 .Ë.....
000100 00000000 00000000 3ECAFBBD C2000000 .....-Û"B... <= address of member
000110 52040000 0000EB80 00010000 00060000 ê.....00.....
000120 D911A9DB A5000400 00000039 40000000 R.zûv.....

Address 3ECAFBBD C2 000000
000000 00010008 00898000 3ECAFBBD C2000000 .....i0...-Û"B...
000010 40010000 00000000 3ECAFBBD C2000830 .....-Û"B...
000020 80000D50 E3D6D7C3 40404040 4040E3D6 0..&TOPC TO
000030 D7C34040 40404040 40404040 40404040 PC
000040 40408000 000007D0 00000008 3F104100 0.....}.....
000050 808C50FA E3AF8000 0E6792A4 C7000000 00&³T®0..ÄkuG...
000060 00000000 00000000 173A0D5C 03000000 .....*.....
000070 3ECAFBBD C2000000 00020000 03000000 .-Û"B.....
000080 814EEC7F 47228000 00000000 00000000 a.0"a.0.....
000090 00000000 00000000 00000000 00000000 .....

```

The ODP is strictly speaking located in the associated space of the cursor objects. We prefer to have our own copy of the pointer to the ODP (we'll use the same pointer in turn for both the ODP of the input file and for the ODP of the output file), so

```
CPYBWP      .ODP, .IFCB-ODP; /* copy pointer to input ODP */
```

Here is the layout of the fixed part of the ODP (note the use of a **SPC**). :

```

DCL SPCPTR .ODP;
DCL SPC      ODP BAS(.ODP);
DCL DD ODP-STATUS      CHAR(4)  DIR;
DCL DD ODP-DEV-LENGTH  BIN(4)   DIR;
DCL DD ODP-OPEN-SIZE   BIN(4)   DIR;
DCL DD ODP-OPEN-FEEDBCK BIN(4)  DIR;
DCL DD ODP.DEV-CTRLBLK BIN(4)  DIR; <=== discussed below
DCL DD ODP.IO-FEEDBACK  BIN(4)  DIR;
DCL DD ODP.LOCK-LIST    BIN(4)  DIR;
DCL DD ODP.SPOOL-OUTPUT BIN(4)  DIR;

DCL DD ODP.MBR-DESCR    BIN(4)  DIR;
DCL DD ODP.CUR-IN-REC   BIN(4)  DIR;
DCL DD ODP.CUR-OUT-REC  BIN(4)  DIR;
DCL DD ODP.OPEN-DMCQ    BIN(4)  DIR;
DCL DD ODP.OUTSTANDINGS BIN(4)  DIR;
DCL DD *                CHAR(12) DIR;

DCL SYSPTR .ODP-CURSOR      DIR;
DCL DD *                CHAR(16) DIR;
DCL SPCPTR .ODP-CDM-ERROR   DIR;

```



DCL	SPCPTR	.ODP-INPUT-BUFFER		DIR;
DCL	SPCPTR	.ODP-OUTPUT-BUFFER		DIR;
DCL	DD	ODP.CDM-CLOSING	BIN(2)	DIR;
DCL	DD	ODP-DEV-NAME-IDX	BIN(2)	DIR;
DCL	DD	ODP-NBR-OF-DEVS	BIN(2)	DIR;
DCL	DD	ODP-SEQUENCE-NBR	BIN(4)	DIR;
DCL	DD	ODP-REC-LENGTH	BIN(2)	DIR;
DCL	DD	ODP-REC-LENGTH2	BIN(2)	DIR;
DCL	DD	ODP-NBR-OF-*RDS	BIN(2)	DIR;
DCL	DD	ODP-RELEASE-NBR	BIN(2)	DIR;
DCL	DD	ODP-OPEN-POSN	CHAR(1)	DIR;
DCL	DD	ODP-OVR-REC-LEN	BIN(2)	DIR;
DCL	DD	ODP-COM-DEV-CNT	BIN(2)	DIR;
DCL	DD	ODP.INPUT-BPCA	BIN(4)	DIR;
DCL	DD	ODP.OUTPUT-BPCA	BIN(4)	DIR;
... stuff omitted				

Looking at the ODP with the Debugger shows:

Variable	Type	Length	Value	ODP	CHARACTER
*	+	1	*	+	1
81000000000008A0000009B0000000B0			a	μ	Λ
00000140000001C60000028000000000			F	Ø	
000002C000000000000000000000002E0			{	\	
000000000000000000000000000000000			Ø	öüþ>	
0000800000000000CFE48EE96E000DFF			°	d	
000005100000000200000490000000000			DBTOPC	LSVA	
8000000000000000EA12C952D364DC10			LGAARD	d	
8000000000000000CFE48EE96E000E60			TOPC	±	
000000000000000000000000000000000			R NØ	¼A	
01900000000000000000000008400000000				±	
2C000000000000000007F000000000000			<		
C4C2E3D6D7C34040404040D3E2E5C1			DATABASE		
D3C7C1C1D9C4000000000000000000000					
000000000000000000000000000840000					
E3D6D7C34040404040400000008F0000					
00000015000000000000000000004B7C1					
D900D5800000000000000000000000000					
000000000000000000000000010000008F00					
000430000000000000000000000000001					
0000000100004C0000FFFF00000000000					
00010001C4C1E3C1C2C1E2C540400000					
000000000000000000010000E00450045					
0045004500450045006F004500450045					
00450BFD00450045000D001100000001					
000000000000000000000000000000000					
000000000000000000000000000000000					
00010001C4C1E3C1C2C1E2C540400000					
000000000000000000010000E00450045					
0045004500450045006F004500450045					
00450BFD00450045000D001100000001					
000000000000000000000000000000000					
000000000000000000000000000000000					

## Data Management Entry Point Table

Here is not the place for a discussion of all the fields of the ODP. What we are interesting in at this point is the Data Management Entry Point table. The **ODP.DEV-CTRLBLK** is an offset to what we need. To get a pointer to this area we add the offset to the pointer to the ODP:

```
ADDSP      .DEV-CONTROL-BLOCK, .ODP, ODP.DEV-CTRLBLK;
```

Below is the format of the *Device Control Block* (values from the above dump are shown as */\* comments \*/*). The DMEPT starts 24 bytes into the DCB:

```

DCL SPCPTR .DEV-CONTROL-BLOCK;
DCL SPC DEV-CONTROL-BLOCK BAS(.DEV-CONTROL-BLOCK);
DCL DD DCB-MAX-NBR-OF-DEVICES BIN( 2) DIR; /* 0001 */
DCL DD DCB-DEVICES-IN-THE-ODP BIN( 2) DIR; /* 0001 */
DCL DD DCB-DEVICE-NAME CHAR(10) DIR; /* DATABASE */
DCL DD DCB-OFFSET-TO-FM-WORK BIN( 4) DIR; /* 00000000 */
DCL DD DCB-LENGTH-OF-FM-WORK BIN( 4) DIR; /* 00000000 */
DCL DD DCB-INDEX-FOR-LUD-PTR BIN( 2) DIR; /* 0000 */
/* DMEPT starts here */
DCL DD DCB-GET BIN( 2) DIR; /* 0010 = 16 QDBGETSQ */
DCL DD DCB-GET-BY-RRN BIN( 2) DIR; /* 000E = 14 QDBGETQD */
DCL DD DCB-GET-BY-KEY BIN( 2) DIR; /* 0045 = 69 QDMIFERR */

```



```

DCL DD *                               BIN( 2) DIR; /* 0045 = 69 QDMIFERR */
DCL DD DCB-PUT                         BIN( 2) DIR; /* 0045 = 69 QDMIFERR */
DCL DD DCB-PUT-GET                     BIN( 2) DIR; /* 0045 = 69 QDMIFERR */
DCL DD DCB-UPDATE                      BIN( 2) DIR; /* 0045 = 69 QDMIFERR */
DCL DD DCB-DELETE                      BIN( 2) DIR; /* 0045 = 69 QDMIFERR */
DCL DD DCB-FORCE-EOD                   BIN( 2) DIR; /* 006F = 111 QDBFEOD */
DCL DD DCB-FORCE-EOV                   BIN( 2) DIR; /* 0045 = 69 QDMIFERR */
DCL DD DCB-COMMIT                       BIN( 2) DIR; /* 0045 = 69 QDMIFERR */
DCL DD DCB-ROLLBACK                     BIN( 2) DIR; /* 0045 = 69 QDMIFERR */
DCL DD DCB-FREE-REC-LOCK                 BIN( 2) DIR; /* 0045 = 69 QDMIFERR */
DCL DD *                               BIN( 2) DIR; /* 0BFD = 3069 QDBCHEOD */
DCL DD *                               BIN( 2) DIR; /* 0045 = 69 QDMIFERR */
DCL DD *                               BIN( 2) DIR; /* 0045 = 69 QDMIFERR */
DCL DD DCB-CLOSE                       BIN( 2) DIR; /* 000D = 13 QDBCLOSE */
DCL DD DCB-OPEN                         BIN( 2) DIR; /* 0011 = 17 QDBOPEN */
DCL DD *                               BIN( 2) DIR; /* 0000 */
DCL DD *                               BIN( 2) DIR; /* 0001 */

```

Each entry in the DMEPT is set to an index or subscript into the SEPT at open time. If an entry does not apply to a particular device/file type, these entry points within the SEPT will be to a program that will signal an appropriate exception saying that the requested operation is not supported or applicable for that device. As you can see, all the output operations for this open input file lead to the same error routine - SEPT entry number 69, QDMIFERR. We are interested in the read-routine, which is SEPT entry number 16 - QDBGETSQ. We save that SEPT entry number for later use:

```
CPYNV          GET-ENTRY, DCB-GET;
```

Similar considerations apply for the output file, so opening the two files now proceeds like this:

```
DCL DD GET-ENTRY BIN(2);
DCL DD PUT-ENTRY BIN(2);
```

OPEN-INPUT-FILE:

```

...
CPYBWP      .INBUF, .IFCB-INBUF;
CPYBWP      .ODP-ROOT, .IFCB-ODP;
ADDSPP      .DEV-CONTROL-BLOCK, .ODP-ROOT, ODP.DEV-NAMELIST;
CPYNV       GET-ENTRY, DCB-GET;

```

OPEN-OUTPUT-FILE:

```

...
CPYBWP      .OUTBUF, .OFCB-OUTBUF;
CPYBWP      .ODP-ROOT, .OFCB-ODP;
ADDSPP      .DEV-CONTROL-BLOCK, .ODP-ROOT, ODP.DEV-NAMELIST;
CPYNV       PUT-ENTRY, DCB-PUT;

```

## I/O Routines

All OS/400 CPF-level I/O routines accept three arguments: an open UFCB, an option list, and a control list. An argument may be left out or coded as a **NULL** pointer, if it is not applicable. Let's set up the operand lists for GET and PUT operations.

It would be convenient if we could initialize a null pointer or pass a null pointer as a parameter like this:

```

DCL SPCPTR .NULL INIT(*); /* invalid initialization */
DCL OL GET-OPERATION(.IFCB, .GET-PARM, *); /* invalid operand */

```

but we can't because the MI-compiler complains, so we have to resort to the following method:

```

DCL SPCPTR .NULL;
DCL OL GET-OPERATION(.IFCB, .GET-OPTION, .NULL);
CPYBWP .NULL, *; /* Make NULL ptr at runtime */

```

## The Data Management Option List

The data management option list is passed to the I/O programs as a space pointer that points to a four-byte option list, which in turn is a fixed structure that has four one-byte substructures.

Operation option byte (byte 0):

```

x'00'   Release All
x'01'   Release First

```

x'01'	Get First
x'02'	Get Last
x'03'	Get Next, Wait
x'04'	Get Previous
x'05'	GetK Next, Unique
x'06'	GetK Previous, Unique
x'07'	GetD RelCur
x'08'	GetD Ordinal RelStr, wait
x'13'	Get Next, No Wait
x'23'	Get Next, Wait via Acc Input
x'83'	Get Next, Wait via Event Handler
x'0C'	Get Cancel
x'0E'	Get Same
x'18'	GetD Ordinal RelStr, No Wait
x'0A'	GetD Next Mod, Wait
x'1A'	GetD Next Mod, No Wait
x'0B'	GetD Unblocked, Wait
x'1B'	GetD Unblocked, No Wait
x'0C'	GetD Cancel
x'0B'	GetK Key Equal
x'0A'	GetK Key Before or Equal
x'09'	GetK Key Before
x'0C'	GetK Key After or Equal
x'0D'	GetK Key After
x'00'	Put, Wait
x'10'	Put, No Wait
x'03'	PutGet Next, Wait
x'13'	PutGet Next, No Wait
x'0C'	PutGet Cancel
x'0F'	Put Send Error Message

Share option byte (byte 1):

x'00'	Get for Read Only, Normal
x'03'	Get for Update, Normal
x'10'	Get for Read Only, No Position
x'13'	Get for Update, No Position

Data option byte (byte 2):

x'00'	Data Access Record
x'01'	No Data Access Record
x'02'	Data Access All Record
x'03'	No Data Access All Record
x'10'	Get Prior, Data Access Record
x'11'	Get Prior, No Data Access Record
x'12'	Get Prior, Data Access All Record
x'13'	Get Prior, No Data Access All Record

Device Support byte (byte 3):

x'01'	Indicating a Get Request
x'02'	Indicating a GetD Request
x'05'	Indicating a Put Request
x'06'	Indicating a PutGet Request
x'07'	Indicating an Update Request
x'08'	Indicating a Delete Request
x'0B'	Indicating a Release Request

So, based on all that, we set up the option lists as follows:



```

CPYBWP      .CMPR-OUTPUT, .OUTPUT-SPACE;      /* set output area      */
CPRDATA     .COMPRESS;                        /* compress the data     */

```

We have chosen the popular LZ1 algorithm (also called LZ77 after Lempel and Ziv's 1977 landmark paper). The LZ77 algorithm is used in many popular compression packages, such as PKZip, WinZip, PNG, and ARJ. In the LZ77 approach, the dictionary is simply a portion of the previously encoded sequence. The encoder examines the input sequence through a sliding window. The window consists of two parts, a *search buffer* that contains a portion of the recently encoded sequence, and a *look-ahead* buffer that contains the next portion of the sequence to be encoded.

To encode the sequence in the look-ahead buffer, the encoder moves a search pointer back through the search buffer until it encounters a match to the first symbol in the look-ahead buffer. The distance of the pointer from the look-ahead buffer is called the *offset*. The encoder then examines the symbols at the pointer location to see if they match consecutive symbols in the look-ahead buffer. The number of consecutive symbols in the search buffer that match consecutive symbols in the look-ahead buffer, starting with the first symbol, is called the *length* of the match. Once the longest match has been found, the encoder encodes it with a triple {*offset*, *length*, *code*} where the code is the codeword corresponding to the symbol in the look-ahead buffer.

## Write the Compressed Records

When outputting the compressed data, we'll precede each compressed chunk by a header record, that contains information about the file and of the size of the current chunk:

```

DCL SPCPTR .OUTBUF;
DCL DD OUTBUF CHAR(132) BAS(.OUTBUF);
DCL DD OUTBUF-SYSTEM CHAR(10) DEF(OUTBUF) POS( 1);
DCL DD OUTBUF-LIB CHAR(10) DEF(OUTBUF) POS(12);
DCL DD OUTBUF-FILE CHAR(10) DEF(OUTBUF) POS(23);
DCL DD OUTBUF-BYTES ZND(10,0) DEF(OUTBUF) POS(34);
DCL DD OUTBUF-RECS ZND(10,0) DEF(OUTBUF) POS(45);

WRITE-HEADER-RECORD:
CPYBREP OUTBUF, " ";
CPYBLA OUTBUF-SYSTEM, NWA-SYSNAME; /* the name of the AS/400 system */
CPYBLA OUTBUF-LIB, IFCB-LIBRARY; /* name of library for input file */
CPYBLA OUTBUF-FILE, IFCB-FILE; /* name of file being compressed */

CPYNV OUTBUF-BYTES, CMPR-ACTUAL-LENGTH; /* number of bytes in chunk */
ADDN OUTPUT-BYTES, CMPR-ACTUAL-LENGTH, 131; /* prepare to round up */
DIV OUTPUT-RECS, OUTPUT-BYTES, 132; /* compute number of records */

CPYNV OUTBUF-RECS, OUTPUT-RECS;
CALLX .SEPT(PUT-ENTRY), PUT-OPERATION, *; /* write header record */
CPYNV CURRENT-REC, 0; /* reset record counter */

WRITE-COMPRESSED-RECORD:
ADDN(S) CURRENT-REC, 1;
CPYBLA OUTBUF, OUTPUT-SPACE(CURRENT-REC);
CALLX .SEPT(PUT-ENTRY), PUT-OPERATION, *;
SUBN(SB) OUTPUT-RECS, 1/NZER(WRITE-COMPRESSED-RECORD);

```

If we are not yet at End-Of-File we go back and read the next batch of records to compress:

```

CMPBLA(B) INPUT-EOF, "Y"/NEQ(READ-UNCOMPRESSED-RECORD);

```

Otherwise go close the files and exit the program:

```

B CLOSE-ALL-FILES;
...
CLOSE-ALL-FILES:
CALLX .SEPT(CLOSE-ENTRY), CLOSE-I, *;
CALLX .SEPT(CLOSE-ENTRY), CLOSE-O, *;
RTX *;

```

## Getting the System Name

There is a handy API, **QWCRNETA**, that we can use to retrieve the name of the local system in which you are running. It is entry number 4938 in the SEPT. We call it like this (after having defined the operand list as shown below):

```
CALLX      .SEPT(4938), QWCRNETA, *;
```

First the Network Attribute Template:

```
DCL SPCPTR .NETWORK-ATTR INIT(NETWORK-ATTR);
DCL DD     NETWORK-ATTR CHAR(32);
DCL DD     NETWORK-ATTR-NBR      BIN(4)  DEF(NETWORK-ATTR) POS( 1);
DCL DD     NETWORK-ATTR-OFFSET   BIN(4)  DEF(NETWORK-ATTR) POS( 5);
DCL DD     NWA-ATTR-NAME         CHAR(10) DEF(NETWORK-ATTR) POS( 9);
DCL DD     NWA-ATTR-TYPE         CHAR(1)  DEF(NETWORK-ATTR) POS(19);
DCL DD     NWA-ATTR-STS          CHAR(1)  DEF(NETWORK-ATTR) POS(20);
DCL DD     NWA-ATTR-SIZE         BIN(4)  DEF(NETWORK-ATTR) POS(21);
DCL DD     NWA-SYSNAME          CHAR(8)  DEF(NETWORK-ATTR) POS(25);

DCL SPCPTR .LENGTH-NETWORK-ATTR INIT(LENGTH-NETWORK-ATTR);
DCL DD     LENGTH-NETWORK-ATTR BIN(4) INIT(32);

DCL SPCPTR .NBR-OF-NETWORK-ATTR INIT(NBR-OF-NETWORK-ATTR);
DCL DD     NBR-OF-NETWORK-ATTR BIN(4) INIT(1);

DCL SPCPTR .NAME-NETWORK-ATTR INIT(NAME-NETWORK-ATTR);
DCL DD     NAME-NETWORK-ATTR CHAR(10) INIT("SYSNAME");

DCL SPCPTR .ERROR-CODE INIT(ERROR-CODE);
DCL DD     ERROR-CODE BIN(4) INIT(0); /* if 0, don't handle errors */

DCL OL QWCRNETA(.NETWORK-ATTR,
               .LENGTH-NETWORK-ATTR,
               .NBR-OF-NETWORK-ATTR,
               .NAME-NETWORK-ATTR,
               .ERROR-CODE);
```

## Decompressing the Data

Decompression consists of reading the compressed chunks and decompressing each one, writing the decompressed data back out. We first read a header record, then as many records as it says in the header, decompress the data with the “Decompress Data”-instruction, **DCPDATA**, write out the result, then read the next header record, etc, until all the data has been processed. The code is straightforward:

### DECOMPRESS-FILE:

```
READ-COMPRESSED-HEADER:
CALLX      .SEPT(GET-ENTRY), GET-OPERATION, *;
CPYINV(B)  NBR-OF-RECS, INBUF-RECS/EQ(CLOSE-ALL-FILES);
CPYINV     NBR-OF-BYTES, INBUF-BYTES;
CPYINV     INPUT-RECS, 0;

READ-COMPRESSED-RECORD:
CMPNV(B)   INPUT-RECS, NBR-OF-RECS/EQ(DECOMPRESS-CHUNK);
CALLX      .SEPT(GET-ENTRY), GET-OPERATION, *;
ADDN(S)    INPUT-RECS, 1;
CPYBLA     INPUT-SPACE(INPUT-RECS), INBUF;
B          READ-COMPRESSED-RECORD;

DECOMPRESS-CHUNK:
CPYINV     CMPR-INPUT-LENGTH, 0;
CPYINV     CMPR-OUTPUT-LENGTH, 660000;
CPYINV     CMPR-ALGORITHM, 0;
CPYBWP     .CMPR-INPUT, .INPUT-SPACE;
CPYBWP     .CMPR-OUTPUT, .OUTPUT-SPACE;
DCPDATA    .COMPRESS; /* uses same template as CPRDATA */

DIV        OUTPUT-RECS, CMPR-ACTUAL-LENGTH, 132;
CPYINV     CURRENT-REC, 0;

WRITE-UNCOMPRESSED-RECORD:
ADDN(S)    CURRENT-REC, 1;
CPYBLA     OUTBUF, OUTPUT-SPACE(CURRENT-REC);
CALLX      .SEPT(PUT-ENTRY), PUT-OPERATION, *;
SUBN(SB)   OUTPUT-RECS, 1/NZER(WRITE-UNCOMPRESSED-RECORD);
B          READ-COMPRESSED-HEADER;
```

## Compression Ratio

A typical listing file had 1207 records totaling 172,032 bytes; the compressed file had 233 records totaling 40960 bytes. The compression ratio was thus  $172032/40960 = 4.2$ , which is typical for listing text.

## The Complete Program

Here is then the complete **MIFCMPS** program:

```
DCL SPCPTR .PARM1 PARM;
DCL DD PARM-OPERATION CHAR(1) BAS(.PARM1);

DCL SPCPTR .PARM2 PARM;
DCL DD PARM-IN-FILE CHAR(10) BAS(.PARM2);

DCL SPCPTR .PARM3 PARM;
DCL DD PARM-IN-LIB CHAR(10) BAS(.PARM3);

DCL SPCPTR .PARM4 PARM;
DCL DD PARM-OUT-FILE CHAR(10) BAS(.PARM4);

DCL SPCPTR .PARM5 PARM;
DCL DD PARM-OUT-LIB CHAR(10) BAS(.PARM5);

DCL OL PARMS(.PARM1, .PARM2, .PARM3, .PARM4, .PARM5) EXT PARM MIN(5);

DCL SPCPTR .INPUT-SPACE INIT(INPUT-SPACE);
DCL DD INPUT-SPACE (5000) CHAR(132) BDRY(16);

DCL SPCPTR .OUTPUT-SPACE INIT(OUTPUT-SPACE);
DCL DD OUTPUT-SPACE (5000) CHAR(132) BDRY(16);

DCL SPCPTR .COMPRESS INIT(COMPRESS);
DCL DD COMPRESS CHAR(64) BDRY(16);
DCL DD CMPR-INPUT-LENGTH BIN(4) DEF(COMPRESS) POS( 1);
DCL DD CMPR-OUTPUT-LENGTH BIN(4) DEF(COMPRESS) POS( 5);
DCL DD CMPR-ACTUAL-LENGTH BIN(4) DEF(COMPRESS) POS( 9);
DCL DD CMPR-ALGORITHM BIN(2) DEF(COMPRESS) POS(13);
DCL DD * CHAR(18) DEF(COMPRESS) POS(15);
DCL SPCPTR .CMPR-INPUT DEF(COMPRESS) POS(33);
DCL SPCPTR .CMPR-OUTPUT DEF(COMPRESS) POS(49);

DCL SPCPTR .ODP;
DCL SPC ODP BAS(.ODP);
DCL DD ODP-STATUS CHAR(4) DIR;
DCL DD ODP-DEV-LENGTH BIN(4) DIR;
DCL DD ODP-OPEN-SIZE BIN(4) DIR;
DCL DD ODP.OPEN-FEEDBCK BIN(4) DIR;
DCL DD ODP.DEV-CTRLBLK BIN(4) DIR;
DCL DD ODP.IO-FEEDBACK BIN(4) DIR;
DCL DD ODP.LOCK-LIST BIN(4) DIR;
DCL DD ODP.SPOOL-OUTPUT BIN(4) DIR;

DCL DD ODP.MBR-DESCR BIN(4) DIR;
DCL DD ODP.CUR-IN-REC BIN(4) DIR;
DCL DD ODP.CUR-OUT-REC BIN(4) DIR;
DCL DD ODP.OPEN-DMCQ BIN(4) DIR;
DCL DD ODP.OUTSTANDINGS BIN(4) DIR;
DCL DD * CHAR(12) DIR;

DCL SYSPTR .ODP-CURSOR DIR;
DCL SPCPTR * DIR;
DCL SPCPTR .ODP-CDM-ERROR DIR;
DCL SPCPTR .ODP-INPUT-BUFFER DIR;
DCL SPCPTR .ODP-OUTPUT-BUFFER DIR;

DCL DD ODP.CDM-CLOSING BIN(2) DIR;
DCL DD ODP-DEV-NAME-IDX BIN(2) DIR;
DCL DD ODP-NBR-OF-DEVS BIN(2) DIR;

DCL DD ODP-SEQUENCE-NBR BIN(4) DIR;
DCL DD ODP-REC-LENGTH BIN(2) DIR;
DCL DD ODP-REC-LENGTH2 BIN(2) DIR;
DCL DD ODP-NBR-OF-*RDS BIN(2) DIR;
DCL DD ODP-RELEASE-NBR BIN(2) DIR;
DCL DD ODP-OPEN-POSN CHAR(1) DIR;
DCL DD ODP-OVR-REC-LEN BIN(2) DIR;
DCL DD ODP-COM-DEV-CNT BIN(2) DIR;
```

```

DCL DD ODP.INPUT-BPCA    BIN(4)    DIR;
DCL DD ODP.OUTPUT-BPCA   BIN(4)    DIR;
DCL DD ODP.....         CHAR(1)   DIR;

DCL SPCPTR .DEV-CONTROL-BLOCK;
DCL SPC     DEV-CONTROL-BLOCK BAS(.DEV-CONTROL-BLOCK);
DCL DD     DCB-MAX-NBR-OF-DEVICES    BIN( 2) DIR;
DCL DD     DCB-DEVICES-IN-THE-ODP    BIN( 2) DIR;
DCL DD     DCB-DEVICE-NAME           CHAR(10) DIR;
DCL DD     DCB-OFFSET-TO-FM-WORK      BIN( 4) DIR;
DCL DD     DCB-LENGTH-OF-FM-WORK      BIN( 4) DIR;
DCL DD     DCB-INDEX-FOR-LUD-PTR      BIN( 2) DIR;
DCL DD     DCB-GET                    BIN( 2) DIR;
DCL DD     DCB-GET-BY-RRN              BIN( 2) DIR;
DCL DD     DCB-GET-BY-KEY              BIN( 2) DIR;
DCL DD     *                          BIN( 2) DIR;
DCL DD     DCB-PUT                     BIN( 2) DIR;
DCL DD     DCB-PUT-GET                 BIN( 2) DIR;
DCL DD     DCB-UPDATE                  BIN( 2) DIR;
DCL DD     DCB-DELETE                  BIN( 2) DIR;
DCL DD     DCB-FORCE-EOD               BIN( 2) DIR;
DCL DD     DCB-FORCE-EOV               BIN( 2) DIR;
DCL DD     DCB-COMMIT                  BIN( 2) DIR;
DCL DD     DCB-ROLLBACK                 BIN( 2) DIR;
DCL DD     DCB-FREE-REC-LOCK            BIN( 2) DIR;
DCL DD     *                          BIN( 2) DIR;
DCL DD     *                          BIN( 2) DIR;
DCL DD     *                          BIN( 2) DIR;
DCL DD     DCB-CLOSE                   BIN( 2) DIR;
DCL DD     DCB-OPEN                    BIN( 2) DIR;
DCL DD     *                          BIN( 2) DIR;
DCL DD     *                          BIN( 2) DIR;

/* THE I/O IS DONE BY USING THE CALLX INSTRUCTION REFERENCING */
/* A SYSTEM POINTER THAT IS OBTAINED FROM THE ENTRY POINT */
/* TABLE. THE ENTRY POINT TABLE CONTAINS PRE-RESOLVED SYSTEM */
/* POINTERS (THOUSANDS...). THE SYSTEM ENTRY POINT TABLE */
/* IS ADDRESSED BY THE POINTER BASED ON THE PROCESS COMMUNI- */
/* CATION OBJECT (PCO): */
/* PCO POINTER --> POINTER TO SEPT --> PTR TO OS FUNCTION 1 */
/* PTR TO OS FUNCTION 2 */
/* ... */

DCL SYSPTR .SEPT(6440) BAS(@SEPT);
DCL DD PCO CHAR(80) BASPCO;
DCL SPCPTR @SEPT DEF(PCO) POS( 1);

/* THE USER FILE CONTROL BLOCK (UFCB) DEFINES THE FILE NAME, */
/* BUFFER SPACES AND ALL NECESSARY CONTROL INFORMATION NEEDED */
/* TO MANAGE THE FILE. IT ALSO PROVIDES THE FEEDBACKS NEEDED */
/* TO ACCESS VARIOUS STRUCTURES, SUCH AS THE ODP (THE OPEN */
/* DATA PATH). */

DCL DD IFCB CHAR(214) BDRY(16);
DCL SPCPTR .IFCB-ODP                DEF(IFCB) POS( 1);
DCL SPCPTR .IFCB-INBUF              DEF(IFCB) POS( 17);
DCL SPCPTR .IFCB-OUTBUF             DEF(IFCB) POS( 33);
DCL SPCPTR .IFCB-OPEN-FEEDBACK      DEF(IFCB) POS( 49);
DCL SPCPTR .IFCB-IO-FEEDBACK        DEF(IFCB) POS( 65);
DCL SPCPTR .IFCB-NEXT-UFCB          DEF(IFCB) POS( 81);

DCL DD *                            CHAR(32) DEF(IFCB) POS( 97);
DCL DD IFCB-FILE                    CHAR(10) DEF(IFCB) POS(129);
DCL DD IFCB-LIB-ID                  BIN(2)   DEF(IFCB) POS(139);
DCL DD IFCB-LIBRARY                 CHAR(10) DEF(IFCB) POS(141);
DCL DD IFCB-MBR-ID                  BIN(2)   DEF(IFCB) POS(151);
DCL DD IFCB-MEMBER                  CHAR(10) DEF(IFCB) POS(153);

DCL DD IFCB-DEVICE-NAME              CHAR(10) DEF(IFCB) POS(163);
DCL DD IFCB-DEVICE-INDEX             BIN(2)   DEF(IFCB) POS(173);

DCL DD IFCB-FLAGS-1                  CHAR(1)   DEF(IFCB) POS(175) INIT(X'80');
DCL DD IFCB-FLAGS-2                  CHAR(1)   DEF(IFCB) POS(176) INIT(X'20');

DCL DD IFCB-REL-VERSION               CHAR(4)   DEF(IFCB) POS(177);
DCL DD IFCB-INV-MK-COUNT              BIN( 4)   DEF(IFCB) POS(181);
DCL DD IFCB-MORE-FLAGS                CHAR(1)   DEF(IFCB) POS(185);
DCL DD *                            CHAR(23)   DEF(IFCB) POS(186);

DCL DD IFCB-LENGTH-ID                BIN( 2)   DEF(IFCB) POS(209) INIT(1);

```

```

DCL DD IFCB-RECORD-LENGTH BIN (2) DEF(IFCB) POS(211) INIT(132);
DCL DD IFCB-NO-MORE-PARMS BIN (2) DEF(IFCB) POS(213) INIT(32767);

DCL SPCPTR .IFCB INIT(IFCB);
DCL OL OPEN-I(.IFCB);
DCL OL CLOSE-I(.IFCB);

DCL DD OFCB CHAR(214) BDRY(16);
DCL SPCPTR .OFCB-ODP DEF(OFCB) POS( 1);
DCL SPCPTR .OFCB-INBUF DEF(OFCB) POS( 17);
DCL SPCPTR .OFCB-OUTBUF DEF(OFCB) POS( 33);
DCL SPCPTR .OFCB-OPEN-FEEDBACK DEF(OFCB) POS( 49);
DCL SPCPTR .OFCB-IO-FEEDBACK DEF(OFCB) POS( 65);
DCL SPCPTR .OFCB-NEXT-UFCB DEF(OFCB) POS( 81);

DCL DD * CHAR(32) DEF(OFCB) POS( 97);
DCL DD OFCB-FILE CHAR(10) DEF(OFCB) POS(129);
DCL DD OFCB-LIB-ID BIN(2) DEF(OFCB) POS(139);
DCL DD OFCB-LIBRARY CHAR(10) DEF(OFCB) POS(141);
DCL DD OFCB-MBR-ID BIN(2) DEF(OFCB) POS(151);
DCL DD OFCB-MEMBER CHAR(10) DEF(OFCB) POS(153);

DCL DD OFCB-DEVICE-NAME CHAR(10) DEF(OFCB) POS(163);
DCL DD OFCB-DEVICE-INDEX BIN(2) DEF(OFCB) POS(173);

DCL DD OFCB-FLAGS-1 CHAR(1) DEF(OFCB) POS(175) INIT(X'80');
DCL DD OFCB-FLAGS-2 CHAR(1) DEF(OFCB) POS(176) INIT(X'10');

DCL DD OFCB-REL-VERSION CHAR(4) DEF(OFCB) POS(177);
DCL DD OFCB-INV-MK-COUNT BIN (4) DEF(OFCB) POS(181);
DCL DD OFCB-MORE-FLAGS CHAR(1) DEF(OFCB) POS(185);
DCL DD * CHAR(23) DEF(OFCB) POS(186);

DCL DD OFCB-LENGTH-ID BIN (2) DEF(OFCB) POS(209) INIT(1);
DCL DD OFCB-RECORD-LENGTH BIN (2) DEF(OFCB) POS(211) INIT(132);
DCL DD OFCB-NO-MORE-PARMS BIN (2) DEF(OFCB) POS(213) INIT(32767);

DCL SPCPTR .OFCB INIT(OFCB);
DCL OL OPEN-O(.OFCB) ARG;
DCL OL CLOSE-O(.OFCB) ARG;

DCL DD GET-ENTRY BIN(2);
DCL DD PUT-ENTRY BIN(2);

DCL CON CLOSE-ENTRY BIN(2) INIT(11);
DCL CON OPEN-ENTRY BIN(2) INIT(12);
DCL CON *LIBL BIN(2) INIT(-75); /* S/38: -72 */
DCL CON *FIRST BIN(2) INIT(-71); /* S/38: -73 */
DCL CON THE-LIB BIN(2) INIT(72);
DCL CON THE-MBR BIN(2) INIT(73);

DCL EXCM * EXCID(H'5001') BP(EOF-DETECTED) CV("CPF") IMD;

DCL SPCPTR .INBUF;
DCL DD INBUF CHAR(132) BAS(.INBUF);
DCL DD INBUF-SYSTEM CHAR(10) DEF(INBUF) POS( 1);
DCL DD INBUF-LIB CHAR(10) DEF(INBUF) POS(12);
DCL DD INBUF-FILE CHAR(10) DEF(INBUF) POS(23);
DCL DD INBUF-BYTES ZND(10,0) DEF(INBUF) POS(34);
DCL DD INBUF-RECS ZND(10,0) DEF(INBUF) POS(45);

DCL SPCPTR .OUTBUF;
DCL DD OUTBUF CHAR(132) BAS(.OUTBUF);
DCL DD OUTBUF-SYSTEM CHAR(10) DEF(OUTBUF) POS( 1);
DCL DD OUTBUF-LIB CHAR(10) DEF(OUTBUF) POS(12);
DCL DD OUTBUF-FILE CHAR(10) DEF(OUTBUF) POS(23);
DCL DD OUTBUF-BYTES ZND(10,0) DEF(OUTBUF) POS(34);
DCL DD OUTBUF-RECS ZND(10,0) DEF(OUTBUF) POS(45);

DCL SPCPTR .NULL;

DCL DD GET-OPTION BIN(4) INIT(H'03000001');
DCL SPCPTR .GET-OPTION INIT(GET-OPTION);
DCL OL GET-OPERATION(.IFCB, .GET-OPTION, .NULL);

DCL DD PUT-OPTION BIN(4) INIT(H'10000005');
DCL SPCPTR .PUT-OPTION INIT(PUT-OPTION);
DCL OL PUT-OPERATION(.OFCB, .PUT-OPTION, .NULL);

DCL DD INPUT-EOF CHAR(1);
DCL DD INPUT-RECS BIN(4);

```



```

DCL DD CURRENT-REC  BIN(4);
DCL DD OUTPUT-RECS  BIN(4);
DCL DD OUTPUT-BYTES BIN(4);
DCL DD NBR-OF-RECS  BIN(4);
DCL DD NBR-OF-BYTES BIN(4);

DCL SPCPTR .NETWORK-ATTR INIT(NETWORK-ATTR);
DCL DD     NETWORK-ATTR CHAR(32);
DCL DD     NETWORK-ATTR-NBR  BIN(4) DEF(NETWORK-ATTR) POS( 1);
DCL DD     NETWORK-ATTR-OFFSET BIN(4) DEF(NETWORK-ATTR) POS( 5);
DCL DD     NWA-ATTR-NAME      CHAR(10) DEF(NETWORK-ATTR) POS( 9);
DCL DD     NWA-ATTR-TYPE      CHAR(1)  DEF(NETWORK-ATTR) POS(19);
DCL DD     NWA-ATTR-STS       CHAR(1)  DEF(NETWORK-ATTR) POS(20);
DCL DD     NWA-ATTR-SIZE      BIN(4)    DEF(NETWORK-ATTR) POS(21);
DCL DD     NWA-SYSNAME        CHAR(8)   DEF(NETWORK-ATTR) POS(25);

DCL SPCPTR .LENGTH-NETWORK-ATTR INIT(LENGTH-NETWORK-ATTR);
DCL DD     LENGTH-NETWORK-ATTR BIN(4) INIT(32);

DCL SPCPTR .NBR-OF-NETWORK-ATTR INIT(NBR-OF-NETWORK-ATTR);
DCL DD     NBR-OF-NETWORK-ATTR BIN(4) INIT(1);

DCL SPCPTR .NAME-NETWORK-ATTR INIT(NAME-NETWORK-ATTR);
DCL DD     NAME-NETWORK-ATTR CHAR(10) INIT("SYSNAME");

DCL SPCPTR .ERROR-CODE INIT(ERROR-CODE);
DCL DD     ERROR-CODE BIN(4) INIT(0);

DCL OL QWCRNETA(.NETWORK-ATTR,
                .LENGTH-NETWORK-ATTR,
                .NBR-OF-NETWORK-ATTR,
                .NAME-NETWORK-ATTR,
                .ERROR-CODE);

/*****/

ENTRY * (PARMS) EXT;
CALLX  .SEPT(4938), QWCRNETA, *;
CPYBWP .NULL, *; /* MAKE NULL PTR */

OPEN-INPUT-FILE:
CPYBLA  IFCB-FILE,    PARM-IN-FILE;
CPYNV   IFCB-LIB-ID,  THE-LIB;
CPYBLA  IFCB-LIBRARY, PARM-IN-LIB;
CPYNV   IFCB-MBR-ID,  THE-MBR;
CPYBLA  IFCB-MEMBER,  PARM-IN-FILE;
CALLX   .SEPT(OPEN-ENTRY), OPEN-I, *;

CPYBWP  .INBUF, .IFCB-INBUF;
CPYBWP  .ODP, .IFCB-ODP;
ADDSPP  .DEV-CONTROL-BLOCK, .ODP, ODP.DEV-CTRLBLK;
CPYNV   GET-ENTRY, DCB-GET;
CPYNV   INPUT-RECS, 0;
CPYBLA  INPUT-EOF, " ";

OPEN-OUTPUT-FILE:
CPYBLA  OFCB-FILE,    PARM-OUT-FILE;
CPYNV   OFCB-LIB-ID,  THE-LIB;
CPYBLA  OFCB-LIBRARY, PARM-OUT-LIB;
CPYNV   OFCB-MBR-ID,  THE-MBR;
CPYBLA  OFCB-MEMBER,  PARM-OUT-FILE;
CALLX   .SEPT(OPEN-ENTRY), OPEN-O, *;

CPYBWP  .OUTBUF, .OFCB-OUTBUF;
CPYBWP  .ODP, .OFCB-ODP;
ADDSPP  .DEV-CONTROL-BLOCK, .ODP, ODP.DEV-CTRLBLK;
CPYNV   PUT-ENTRY, DCB-PUT;

CMPBLA(B)  PARM-OPERATION, "D"/EQ(DECOMPRESS-FILE);

COMPRESS-FILE:
READ-UNCOMPRESSED-RECORD:
CMPNV(B)  INPUT-RECS, 5000/EQ(COMPRESS-CHUNK);
CALLX     .SEPT(GET-ENTRY), GET-OPERATION, *;
ADDN(S)   INPUT-RECS, 1;
CPYBLA    INPUT-SPACE(INPUT-RECS), INBUF;
B         READ-UNCOMPRESSED-RECORD;

EOF-DETECTED:
CPYBLA    INPUT-EOF, "Y";
CMPBLA(B) PARM-OPERATION, "D"/EQ(CLOSE-ALL-FILES);

```

```

COMPRESS-CHUNK:
  MULT      CMPR-INPUT-LENGTH, INPUT-RECS, 132;
  CPYNV     INPUT-RECS, 0;
  CPYNV     CMPR-OUTPUT-LENGTH, 660000;
  CPYNV     CMPR-ALGORITHM, 2;
  CPYBWP    .CMPR-INPUT, .INPUT-SPACE;
  CPYBWP    .CMPR-OUTPUT, .OUTPUT-SPACE;
  CPRDATA   .COMPRESS;

WRITE-HEADER-RECORD:
  CPYBREP    OUTBUF, " ";
  CPYBLA     OUTBUF-SYSTEM, NWA-SYSNAME;
  CPYBLA     OUTBUF-LIB, IFCB-LIBRARY;
  CPYBLA     OUTBUF-FILE, IFCB-FILE;
  CPYNV     OUTBUF-BYTES, CMPR-ACTUAL-LENGTH;

  ADDN      OUTPUT-BYTES, CMPR-ACTUAL-LENGTH, 131;
  DIV       OUTPUT-RECS, OUTPUT-BYTES, 132;
  CPYNV     OUTBUF-RECS, OUTPUT-RECS;
  CALLX     .SEPT(PUT-ENTRY), PUT-OPERATION, *;
  CPYNV     CURRENT-REC, 0;

WRITE-COMPRESSED-RECORD:
  ADDN(S)    CURRENT-REC, 1;
  CPYBLA     OUTBUF, OUTPUT-SPACE(CURRENT-REC);
  CALLX     .SEPT(PUT-ENTRY), PUT-OPERATION, *;
  SUBN(SB)   OUTPUT-RECS, 1/NZER(WRITE-COMPRESSED-RECORD);
  CMPBLA(B)  INPUT-EOF, "Y"/NEQ(READ-UNCOMPRESSED-RECORD);

CLOSE-ALL-FILES:
  CALLX     .SEPT(CLOSE-ENTRY), CLOSE-I, *;
  CALLX     .SEPT(CLOSE-ENTRY), CLOSE-O, *;
  RTX      *;

DECOMPRESS-FILE:
READ-COMPRESSED-HEADER:
  CALLX     .SEPT(GET-ENTRY), GET-OPERATION, *;
  CPYNV(B)   NBR-OF-RECS, INBUF-RECS/EQ(CLOSE-ALL-FILES);
  CPYNV     NBR-OF-BYTES, INBUF-BYTES;
  CPYNV     INPUT-RECS, 0;
READ-COMPRESSED-RECORD:
  CMPNV(B)   INPUT-RECS, NBR-OF-RECS/EQ(DECOMPRESS-CHUNK);
  CALLX     .SEPT(GET-ENTRY), GET-OPERATION, *;
  ADDN(S)    INPUT-RECS, 1;
  CPYBLA     INPUT-SPACE(INPUT-RECS), INBUF;
  B         READ-COMPRESSED-RECORD;

DECOMPRESS-CHUNK:
  CPYNV     CMPR-INPUT-LENGTH, 0;
  CPYNV     CMPR-OUTPUT-LENGTH, 660000;
  CPYNV     CMPR-ALGORITHM, 0;
  CPYBWP    .CMPR-INPUT, .INPUT-SPACE;
  CPYBWP    .CMPR-OUTPUT, .OUTPUT-SPACE;
  DCPDATA   .COMPRESS;

  DIV       OUTPUT-RECS, CMPR-ACTUAL-LENGTH, 132;
  CPYNV     CURRENT-REC, 0;
WRITE-UNCOMPRESSED-RECORD:
  ADDN(S)    CURRENT-REC, 1;
  CPYBLA     OUTBUF, OUTPUT-SPACE(CURRENT-REC);
  CALLX     .SEPT(PUT-ENTRY), PUT-OPERATION, *;
  SUBN(SB)   OUTPUT-RECS, 1/NZER(WRITE-UNCOMPRESSED-RECORD);
  B         READ-COMPRESSED-HEADER;

```

Blank

## Chapter 13

### File Conversion and Hashing

#### **Minimum File Transformation: COPY**

Having examined in some detail how to read/write files with MI, we pose now the following problem: what is the *minimum* amount of code needed to copy one file to another? We basically take the program we developed in Chapter 12 and slash away stuff we don't need (this is called "Samurai programming"). You should be able to quickly [grok](#)<sup>i</sup> the result (**MIFCOPY**):

```
/* common stuff */
DCL SPCPTR .ODP;
DCL SPC ODP BAS(.ODP);
DCL DD ODP.DCB BIN(4) DEF(ODP) POS(17);

DCL SPCPTR .DCB;
DCL SPC DCB BAS(.DCB);
DCL DD DCB-GET BIN(2) DEF(DCB) POS(25);
DCL DD DCB-PUT BIN(2) DEF(DCB) POS(33);

DCL SPCPTR .NULL;
DCL SPCPTR @SEPT BASPCO);
DCL SYSPTR .SEPT(6440) BAS(@SEPT);

DCL CON CLOSE-ENTRY BIN(2) INIT(11);
DCL CON OPEN-ENTRY BIN(2) INIT(12);

/* Input section */
DCL SPCPTR .IFCB INIT(IFCB);
DCL DD IFCB CHAR(214) BDRY(16);
DCL SPCPTR .IFCB-ODP DEF(IFCB) POS( 1);
DCL SPCPTR .IFCB-INBUF DEF(IFCB) POS( 17);
DCL DD IFCB-FILE CHAR(10) DEF(IFCB) POS(129);
DCL DD IFCB-LIB-ID BIN(2) DEF(IFCB) POS(139) INIT(72);
DCL DD IFCB-LIBRARY CHAR(10) DEF(IFCB) POS(141);
DCL DD IFCB-MBR-ID BIN(2) DEF(IFCB) POS(151) INIT(73);
DCL DD IFCB-MEMBER CHAR(10) DEF(IFCB) POS(153);
DCL DD IFCB-FLAGS-1 CHAR(1) DEF(IFCB) POS(175) INIT(X'80');
DCL DD IFCB-FLAGS-2 CHAR(1) DEF(IFCB) POS(176) INIT(X'20');
DCL DD IFCB-LENGTH-ID BIN (2) DEF(IFCB) POS(209) INIT(1);
DCL DD IFCB-RECORD-LENGTH BIN (2) DEF(IFCB) POS(211) INIT(132);
DCL DD IFCB-NO-MORE-PARMS BIN (2) DEF(IFCB) POS(213) INIT(32767);

DCL OL OPEN-I(.IFCB);
DCL OL CLOSE-I(.IFCB);

DCL DD GET-ENTRY BIN(2);
DCL DD INBUF CHAR(132) BAS(.IFCB-INBUF);
DCL DD GET-OPTION BIN(4) INIT(H'03000001');
DCL SPCPTR .GET-OPTION INIT(GET-OPTION);
DCL OL GET-OPERATION(.IFCB, .GET-OPTION, .NULL);

/* Output section */
DCL SPCPTR .OFCB INIT(OFCB);
DCL DD OFCB CHAR(214) BDRY(16);
DCL SPCPTR .OFCB-ODP DEF(OFCB) POS( 1);
DCL SPCPTR .OFCB-OUTBUF DEF(OFCB) POS( 33);
DCL DD OFCB-FILE CHAR(10) DEF(OFCB) POS(129);
DCL DD OFCB-LIB-ID BIN(2) DEF(OFCB) POS(139) INIT(72);
DCL DD OFCB-LIBRARY CHAR(10) DEF(OFCB) POS(141);
DCL DD OFCB-MBR-ID BIN(2) DEF(OFCB) POS(151) INIT(73);
DCL DD OFCB-MEMBER CHAR(10) DEF(OFCB) POS(153);
DCL DD OFCB-FLAGS-1 CHAR(1) DEF(OFCB) POS(175) INIT(X'80');
DCL DD OFCB-FLAGS-2 CHAR(1) DEF(OFCB) POS(176) INIT(X'10');
DCL DD OFCB-LENGTH-ID BIN (2) DEF(OFCB) POS(209) INIT(1);
DCL DD OFCB-RECORD-LENGTH BIN (2) DEF(OFCB) POS(211) INIT(132);
DCL DD OFCB-NO-MORE-PARMS BIN (2) DEF(OFCB) POS(213) INIT(32767);

DCL DD PUT-ENTRY BIN(2);
DCL DD OUTBUF CHAR(132) BAS(.OFCB-OUTBUF);
DCL DD PUT-OPTION BIN(4) INIT(H'10000005');
DCL SPCPTR .PUT-OPTION INIT(PUT-OPTION);
DCL OL PUT-OPERATION(.OFCB, .PUT-OPTION, .NULL);
```

```

DCL OL OPEN-O(.OFCB) ARG;
DCL OL CLOSE-O(.OFCB) ARG;

DCL EXCM * EXCID(H'5001') BP(EOF-DETECTED) CV("CPF") IMD;

/*****
/* trivial parameter declarations omitted ... */
*****/

ENTRY * (PARMS) EXT;
  CPYBWP      .NULL, *; /* MAKE NULL PTR */

OPEN-INPUT-FILE:
  CPYBLA      IFCB-FILE,    PARM-IN-FILE;
  CPYBLA      IFCB-LIBRARY, PARM-IN-LIB;
  CPYBLA      IFCB-MEMBER,  PARM-IN-FILE;
  CALLX      .SEPT(OPEN-ENTRY), OPEN-I, *;

  CPYBWP      .ODP, .IFCB-ODP;
  ADDSPP      .DCB, .ODP, ODP.DCB;
  CPYNV      GET-ENTRY, DCB-GET;

OPEN-OUTPUT-FILE:
  CPYBLA      OFCB-FILE,    PARM-OUT-FILE;
  CPYBLA      OFCB-LIBRARY, PARM-OUT-LIB;
  CPYBLA      OFCB-MEMBER,  PARM-OUT-FILE;
  CALLX      .SEPT(OPEN-ENTRY), OPEN-O, *;

  CPYBWP      .ODP, .OFCB-ODP;
  ADDSPP      .DCB, .ODP, ODP.DCB;
  CPYNV      PUT-ENTRY, DCB-PUT;

TRANSFORM-FILE:
  CALLX      .SEPT(GET-ENTRY), GET-OPERATION, *;
  CPYBLA      OUTBUF, INBUF; /* just copy */
  CALLX      .SEPT(PUT-ENTRY), PUT-OPERATION, *;
  B          TRANSFORM-FILE;

EOF-DETECTED:
  CALLX      .SEPT(CLOSE-ENTRY), CLOSE-I, *;
  CALLX      .SEPT(CLOSE-ENTRY), CLOSE-O, *;
  RTX      *;

```

## Generic File Transformer Command

We'll use a simple generic command (FTRANS/QCMDSRC) to invoke the various programs developed in this chapter:

```

CMD  PROMPT('Transform In File to Out File')
PARM KWD(PGM)      TYPE(*CHAR) LEN(10) PROMPT('Program')
PARM KWD(INFILE)   TYPE(*CHAR) LEN(10) PROMPT('In File')
PARM KWD(INLIB)    TYPE(*CHAR) LEN(10) PROMPT('In Library')
PARM KWD(OUTFILE)  TYPE(*CHAR) LEN(10) PROMPT('Out File')
PARM KWD(OUTLIB)   TYPE(*CHAR) LEN(10) PROMPT('Out Library')
PARM KWD(PARM)     TYPE(*CHAR) LEN(50) PROMPT('General Parameter')

====> CRTCMD CMD(FTRANS) PGM(CLFTRANS) REPLACE(*YES)

```

Here is its Command Processing Program (CLFTRANS/QCLSRC):

```

PGM PARM(&PGM &INFILE &INLIB &OUTFILE &OUTLIB &PARM)
DCL VAR(&PGM)      TYPE(*CHAR) LEN(10)
DCL VAR(&INFILE)   TYPE(*CHAR) LEN(10)
DCL VAR(&INLIB)    TYPE(*CHAR) LEN(10)
DCL VAR(&OUTFILE)  TYPE(*CHAR) LEN(10)
DCL VAR(&OUTLIB)   TYPE(*CHAR) LEN(10)
DCL VAR(&PARM)     TYPE(*CHAR) LEN(10)

DLTF  FILE(&OUTLIB/&OUTFILE)
MSGID(CPF2105) /* FILE NOT FOUND */
CRTPF  FILE(&OUTLIB/&OUTFILE) RCDLEN(132) +
      SIZE(*NOMAX) LVLCHK(*NO)
CALL PGM(&PGM) PARM(&INFILE &INLIB &OUTFILE &OUTLIB &PARM)
ENDPGM

```

Of course, just copying the file is no big deal. It becomes a lot more interesting if we replace the copy instruction with one that somehow transforms the data. This also gives us a chance of introducing a host of new text processing MI-instructions - the real reason we are doing this ☺. We can also split the program in

two halves, one that can read and one that can write. Reminds me of the old joke about the Belgian police (it might have wider applicability): why are there always two cops in a patrolcar? You see: one that can ... ah, well, you get it.

## Translating the Data From a File

A problem that comes up regularly is translating an EBCDIC text file (e.g. a report) to ASCII. There is an EBCDIC -> ASCII table translation table on every AS/400: **QASCII/QSYS** type/subtype x'1906'. From a dump of the table, it is clear that the associated primary space contains the 256-byte conversion table:

Address	38787C9A03	000000			
000000	00010008	00808000	38787C9A	03000000	.....00..i@a....
000010	C0010000	00000000	38787C9A	03000100	{.....i@a....
000020	80001906	D8C1E2C3	C9C94040	40404040	0... <b>QASCII</b>
000030	40404040	40404040	40404040	40404040	
000040	40400040	00000F00	00000008	3F100301	.....
...					
<b>000100</b>	00010203	9C09867F	978D8E0B	0C0D0E0F	...æ.f"pýþ....
000110	10111213	9D850887	1819928F	1C1D1E1F	....e.g..k±....
000120	80818283	840A171B	88898A8B	8C050607	øabcd...hi«»ð...
000130	90911693	94959604	98999A9B	14159E1A	°j.lmno.qrª°..Æ.
000140	20A0A1A2	A3A4A5A6	A7A85B2E	3C282B21	·µ~stuvwxý\$.....
000150	26A9AAAB	ACADAEAF	B0B15D24	2A293B5E	·zj¿ÐYp®^£)....;
000160	2D2FB2B3	B4B5B6B7	B8B97C2C	255F3E3F	..¥.©\$!%&%@..~..
000170	BABBBBCD	BEFBC0C1	C2603A23	40273D22	[ ]~...x{AB-... ..
000180	C3616263	64656667	6869C4C5	C6C7C8C9	C/AAAAAÇNDEFGHI
000190	CA6A6B6C	6D6E6F70	7172CBCC	CDCECFD0	-!,%_>?øÉÊÖöóóó}
0001A0	D17E7374	75767778	797AD2D3	D4D5D6D7	J=ÉÉÍííí`·KLMNOP
0001B0	D8D9DADB	DCDDDEDF	E0E1E2E3	E4E5E6E7	QR'úúúúý\·STUVWX
0001C0	7B414243	44454647	4849E8E9	EAEBECED	#·ââââââçñYZ²ÖÖÖ
0001D0	7D4A4B4C	4D4E4F50	5152EEEF	F0F1F2F3	'¢.<(.·&éêö00123
0001E0	5C9F5354	55565758	595AF4F5	F6F7F8F9	*æèèííííß!456789
0001F0	30313233	34353637	3839FAFB	FCFDFEFF	.....³ÜÜÜ.

So, an obvious strategy would be to resolve to the table and set a space pointer from the resolved system pointer, then base a 256-character translation table on the resulting pointer:

```
DCL SYSPTR .TRANSLATE-TABLE-OBJECT;
DCL SPCPTR .TRANSLATE-TABLE;
DCL DD      TRANSLATE-TABLE CHAR(256) BAS(.TRANSLATE-TABLE);

DCL DD RESOLVE CHAR(34);
DCL DD RESOLVE-TYPE CHAR( 2) DEF(RESOLVE) POS( 1);
DCL DD RESOLVE-NAME CHAR(30) DEF(RESOLVE) POS( 3);
DCL DD RESOLVE-AUTH CHAR( 2) DEF(RESOLVE) POS(33) INIT(X'0000');

/*****/

ENTRY * (PARMS) EXT;
  CPYBWP      .NULL, *, /* MAKE NULL PTR */

RESOLVE-TO-TABLE:
  CPYBLA      RESOLVE-TYPE, X'1906';
  CPYBLAP     RESOLVE-NAME, "QASCII", " ";
  RSLVSP      .TRANSLATE-TABLE-OBJECT, RESOLVE, *, *;
  SETSPFPF    .TRANSLATE-TABLE, .TRANSLATE-TABLE-OBJECT;
```

Unfortunately, IBM has made the system overly restrictive and the SETSPFPF instruction fails with a “protection violation” because the QASCII table is in the system domain. This makes little sense, but such is the way of Big Blue. We have several options, we could make the program a system-state program, we could have a hard-coded version of the table in our program, or we could use our counterfeit pointer technique from chapter 7. Let’s first try to simply go system state. Then we could do the translation as follows (see program **MIFASCII**):

```
TRANSFORM-FILE:
  CALLX      .SEPT(GET-ENTRY), GET-OPERATION, *;
  XLATEWT    OUTBUF, INBUF, TRANSLATE-TABLE;
  CALLX      .SEPT(PUT-ENTRY), PUT-OPERATION, *;
  B          TRANSFORM-FILE;
```

The “Translate With Table”-instruction, **XLATEWT**, does the translation in one go. The syntax is:

## XLATEWT      *Receiver, Source, Table*

The length of the shortest of *Receiver* and *Source* determines the amount of data being translated. Each character value of the *Source* is used as a subscript or index into the *Table* to select the character to be placed in the *Receiver*.

It is saddening that we need to be a system-state program to *read* the system-state table QASCII. The next option is to hard-code the table (note how I declare 16 unnamed (“”) pieces of 16 bytes each so it is easy to initialize the table):

```
DCL DD TRANSLATE-TABLE CHAR(256);
DCL DD *(16) CHAR(16) DEF(TRANSLATE-TABLE) POS(1) INIT(
  X'000102039C09867F978D8E0B0C0D0E0F',
  X'101112139D8508871819928F1C1D1E1F',
  X'80818283840A171B88898A8B8C050607',
  X'909116939495960498999A9B14159E1A',
  X'20A0A1A2A3A4A5A6A7A85B2E3C282B21',
  X'26A9AAABACADAEAFB0B15D242A293B5E',
  X'2D2FB2B3B4B5B6B7B8B97C2C255F3E3F',
  X'BABBBBCDBEBFC0C1C2603A2340273D22',
  X'C3616263646566676869C4C5C6C7C8C9',
  X'CA6A6B6C6D6E6F707172CBCCDCCECFD0',
  X'D17E737475767778797AD2D3D4D5D6D7',
  X'D8D9DADBDCDDDEDFE0E1E2E3E4E5E6E7',
  X'7B414243444546474849E8E9EAEBCED',
  X'7D4A4B4C4D4E4F505152EEFF0F1F2F3',
  X'5C9F535455565758595AF4F5F6F7F8F9',
  X'30313233343536373839FAFBFCFDFEFF');
```

Alternatively, you could use the following syntax, where each piece is prefixed by a position indicator *\*(pos)*:

```
DCL DD TRANSLATE-TABLE CHAR(256) INIT(
  *( 1) X'000102039C09867F978D8E0B0C0D0E0F',
  *(17) X'101112139D8508871819928F1C1D1E1F',
  ...
  *(225) X'5C9F535455565758595AF4F5F6F7F8F9',
  *(241) X'30313233343536373839FAFBFCFDFEFF');
```

But then you’ll have to count to get the correct positions for each piece. Finally, you have to remove (or comment out) the previous attempt of getting a pointer to the table, of course.

## A GREP-Like Utility

On Unix systems there is an extremely useful tool, *grep*, that can find all occurrences in a file of words (and more -- actually, it works with “regular expressions” like ‘wo??.\*’). We’ll not try here (leave it as an exercise for the reader - just kidding) to implement the full regular expression syntax, but instead we’ll solve the simpler problem of finding just words or more generally finding *strings* with **MIFGREP**:

```
...
DCL SPCPTR .PARM5 PARM;
DCL DD PARM-GENERAL CHAR(50) BAS(.PARM5);
...
DCL DD STRING-TO-FIND CHAR(51);
DCL DD STRING-SIZE BIN(2);
DCL DD WHERE BIN(2);

/*****/
...
OPEN-INPUT-FILE: ...
OPEN-OUTPUT-FILE: ...

GET-STRING:
  CPYBLAP STRING-TO-FIND, PARM-GENERAL, " ";
  TRIML STRING-SIZE, STRING-TO-FIND, " ";

TRANSFORM-FILE:
  CALLX .SEPT(GET-ENTRY), GET-OPERATION, *;
  SCAN(B) WHERE, INBUF, STRING-TO-FIND(1:STRING-SIZE)
  /ZER(TRANSFORM-FILE);
  CPYBLA OUTBUF, INBUF;
  CALLX .SEPT(PUT-ENTRY), PUT-OPERATION, *;
  B TRANSFORM-FILE;
```

We use the 5<sup>th</sup> parameter (GENERAL) to hold the string to look for. We need the size of the string up to and including the last non-blank character. The “Trim Length”-instruction, **TRIML**, can be used for that:

```
TRIML  Length, Source, Character ; /* find Length of Source */
```

where *Length* is a numerical variable computed by reducing the length of the *Source* by one for each occurrence of *Character* scanning from the right until a character is found that is not equal to *Character* or the length has been reduced to zero, thus precisely what we need.

Scanning the input buffer for the string can be done with the “Scan”-instruction, **SCAN**, which scans the *Source* string for occurrences of the *Compare* string, recording the position of the *Compare* string *Where* it was found, otherwise setting *Where* to zero:

```
SCAN  where, Source, Compare ; /* scan for Compare string */
```

As compare operand we’ll use a substring of our STRING-TO-FIND variable with a length just determined:

```
TRANSFORM-FILE:
CALLX  .SEPT(GET-ENTRY), GET-OPERATION, *;
SCAN(B)  WHERE, INBUF, STRING-TO-FIND(1:STRING-SIZE)
          /ZER(TRANSFORM-FILE);
```

If **WHERE** returns zero, the string was not found, so we go get the next record. Now, an interesting question is: “what if the size of the string to find is zero?”. Such a string is either everywhere or nowhere depending on your point of view. Sometimes you want to say that you have a match because as the string gets shorter and shorter, the search becomes less and less restrictive (a length of zero being not restrictive at all), and sometimes you want to say “if there is nothing to look for, don’t do anything”. The **SCAN** instruction provides a way of handling this. If the substring is null (*i.e.* its length is zero), the *Where* operand returns zero just as if the compare string was not found, but the branch condition is set to the special value **NCMP** (for Null Compare). Since our code loops back on the zero condition, which is distinct from the null compare condition, it means that we are accepting all records if the string to search for is empty. If we wanted to include only records where a non-empty string was found, we should loop back on a **/NPOS**-condition (not positive) as I do in the program to follow.

## A KWIC (KeyWord-In-Context) Version of GREP

This program, **MIFKWIC**, is a variation of **MIFGREP**. The string we are looking for selects all records that match, but instead of just outputting a file of matching records, we’ll format the output such that each line (corresponding to one *hit*) begins with the record number in the input file, followed by the input record shifted either left or right such that the string, we are matching on, is aligned on a fixed column position (say 61). The only problem here is to calculate the lengths of the portion before the string and of the portion from the string and on to the end. The strategy is then to copy these two portions to two fields of the output buffer, right-adjusted to the first field and left-adjusted to the second field; in both cases padding as needed with blanks:

```
DCL DD STRING-SIZE BIN(2);
DCL DD THE-LENGTH  BIN(2);
DCL DD WHERE       BIN(2);
DCL DD REC-NBR     ZND(10,0);
...
CPYNV          REC-NBR, 0;

TRANSFORM-FILE:
CALLX  .SEPT(GET-ENTRY), GET-OPERATION, *;
ADDN(S) REC-NBR, 1;
SCAN(B)  WHERE, INBUF, STRING-TO-FIND(1:STRING-SIZE)
          /NPOS(TRANSFORM-FILE);

CPYBLAP      OUTBUF, REC-NBR, " "; /* AND CLEAR REST OF RECORD */

SUBN(B)      THE-LENGTH, WHERE, 1/NPOS(=+2); /* BEFORE STRING */
CPYBRAP      OUTBUF(12:49), INBUF( 1:THE-LENGTH), " ";;

SUBN(B)      THE-LENGTH, 133, WHERE/NPOS(=+2); /* FROM STRING ON */
CPYBLAP      OUTBUF(61:72), INBUF(WHERE:THE-LENGTH), " ";;

CALLX  .SEPT(PUT-ENTRY), PUT-OPERATION, *;
B      TRANSFORM-FILE;
```

The “Copy Bytes Right-Adjusted with Pad”-instruction, **CPYBRAP**, nicely does the right-adjusting for us. When I first read the MIFR and came across **CPYBRAP**, I asked myself “why would anyone want to do



that?”. Well here is one application. Another one is copying a zoned number to a field longer than the number, then padding with “0” to get the required number of leading zeroes. Or a binary number, padding with x’00’.

The MIFR states that “Substring operand references that allow for a null substring reference (a length value of zero) may be specified for operands 1 and 2”. However, when I tried it, I got a “subscript out of range” error. We could, of course, suppress that with the **OVPGATR** instruction we learned about in chapter 10, but I would rather not, so the calculation of THE-LENGTH has an explicit test on Not Positive and skips the copy-instruction, if needed:

```
SUBN(B)      THE-LENGTH, WHERE, 1/NPOS(=+2); /* BEFORE STRING */
CPYBRAP      OUTBUF(12:49), INBUF( 1:THE-LENGTH), " ";;
```

Calculation of the second length can never yield a non-positive result, but the test and the structure of the instruction are maintained as for the first to make the code look a bit cleaner. Some people might disagree with this point, but they are of course allowed to modify their code as they see fit.

Here is an example of what the program produces when working on dump of itself with the search string being “VERSION”:

```
0000000063 0100000002008004          PROGRAM VERSION TBL PTR 256C1A4D39 000280
0000000100          TABLE VERSION TABLE
0000000101          TABLE VERSION 00          BN VRM OX LEVEL 02          BN INTERNAL VR
0000000112          TABLE VERSION 00          INTERRUPTED OP 00
0000000301 0018      TABLE ENTRY CNT 00000002          TABLE VERSION 00          RESERVED 0000000000000000
0000000310 0040      TABLE ENTRY CNT 00000001          TABLE VERSION 00          RESERVED 0000000000000000
0000000325          TABLE VERSION 00          MOD TYPE 03          MOD STAT
0000000327          TABLE VERSION TBL PTR 256C1A4D39 000540
0000000353          TABLE VERSION 00          RESERVED 0000000000000000
0000000364          TABLE VERSION TABLE
0000000366          TABLE VERSION 00          MM VRM OX LEVEL 02          LANGUAGE
0000000422          TABLE VERSION 00          PROC TYPE 03          PROC BDY
0000000439          TABLE VERSION 00          BASIC BLK ATTRS 00          RESERVED
0000001549          TABLE VERSION 2          PM OFFSET 0
0000001558          TABLE VERSION 1          SNA PM OFFSET 0
0000001568          TABLE VERSION 2          SNA PM OFFSET 0          BLMV OFF
0000001573          TABLE VERSION 2          NUM FPR SAVED 0          NUM GPR
0000001580          TABLE VERSION 01          RESERVED 0000000000000000
0000001706 0030      TABLE ENTRY CNT 00000002          TABLE VERSION 00          RESERVED 0000000000000000
0000001714 0040      TABLE ENTRY CNT 00000003          TABLE VERSION 00          RESERVED 0000000000000000
0000001739          TABLE VERSION 1          LAST ENTRY IDX 3
0000001747          TABLE VERSION 2          LAST ENTRY IDX 2          RESERVED
0000001766          TABLE VERSION 00          RESERVED 0000000000000000
0000001810          TABLE VERSION 1          LAST ENTRY IDX 46
0000001949 0028      TABLE ENTRY CNT 00000001          TABLE VERSION 00          RESERVED 0000000000000000
0000001959          TABLE VERSION 00          RESERVED 000000          OBS MUST
0000001983          TABLE VERSION 00          RESERVED 0000000000000000
```

## Calculating a File Digest

A file *digest* is a value computed over the entire file with the (ideal) properties that 1) any changes (no matter how small) to the file with very high probability result in a different digest value, and 2) it is unfeasible to produce another file with different contents but with the same digest. [NIST](#), along with the [NSA](#), designed the Secure Hash Algorithm (SHA) for use with the Digital Signature Standard. When a message of any length  $< 2^{64}$  bits is input, the SHA produces a 160-bit output called a message digest. The original SHA specification was later amended to become SHA-1. The change “corrects a technical flaw that made the standard less secure than had been thought”. The NSA has refused to elaborate on the exact nature of the flaw. There are no known successful cryptographic attacks against SHA-1. SHA-1 doesn’t encrypt the file, the message digest produced by SHA-1 on a file, is in effect a ‘unique’ *signature* of that file that can be used by the sender of data to give to the receiver along with the file, so that the receiver can verify the contents of the file have remained untouched while in transit.

The “Cipher”-instruction, **CIPHER**, has built-in support for SHA-1 (and for another digest method called MD5). Let’s look at how the CIPHER instruction works. Here is the format:

```
CIPHER .Receiver, Control, .Source; /* Cipher Source into Receiver under Control */
```

The *.Receiver* and *.Source* operands are space pointers to data areas, while *Control* is a 32-character control block with this format:

```
DCL DD CTRL CHAR(32) BDRY(16);
DCL DD CTRL-FUNCTION CHAR( 2) DEF(CTRL) POS( 1) INIT(X'0005'); /* hash */
DCL DD CTRL-HASH-ALG CHAR( 1) DEF(CTRL) POS( 3) INIT(X'01'); /* SHA-1 */
DCL DD CTRL-SEQUENCE CHAR( 1) DEF(CTRL) POS( 4);
DCL DD CTRL-LENGTH BIN( 4) DEF(CTRL) POS( 5) INIT(132);
DCL DD * CHAR( 8) DEF(CTRL) POS( 9);
DCL SPCPTR .CONTEXT DEF(CTRL) POS(17) INIT(CONTEXT);
```

The .CONTEXT pointer points to a 96-character work-area that is used by the instruction:

```
DCL DD CONTEXT CHAR(96); /* PRIVATE CONTEXT FOR CIPHER INSTRUCTION */
```

A hash calculation of data can be performed in one execution of the CIPHER instruction or in several, which allows the hash to be calculated as we read records from the input file. This is specified using the SEQUENCE field in the control area. When performing the hash calculation in one execution, the SEQUENCE field should specify *only* (X'00'). When performing the hash on each record as we go the first use of the CIPHER instruction should specify *first* (X'01'), the last use of CIPHER should specify *final* (X'03'), and any executions in between should specify *middle* (X'02'). The work-area should be initialized to binary zeroes and must not be changed by the running program between executions of CIPHER. The hash will be returned in the *Receiver* operand when the SEQUENCE field specifies *only* or *final*. This somewhat cramped logic is implemented by:

```
DCL SPCPTR .CIPHER-RESULT INIT(CIPHER-RESULT);
DCL DD      CIPHER-RESULT CHAR(20); /* 160-bit result */

DCL SPCPTR .CIPHER-SOURCE INIT(CIPHER-SOURCE);
DCL DD      CIPHER-SOURCE CHAR(132);

DCL DD CTRL CHAR(32) BDRY(16);
DCL DD CTRL-FUNCTION CHAR( 2) DEF(CTRL) POS( 1) INIT(X'0005');
DCL DD CTRL-HASH-ALG CHAR( 1) DEF(CTRL) POS( 3) INIT(X'01');
DCL DD CTRL-SEQUENCE CHAR( 1) DEF(CTRL) POS( 4);
DCL DD CTRL-LENGTH BIN( 4) DEF(CTRL) POS( 5) INIT(132);
DCL DD * CHAR( 8) DEF(CTRL) POS( 9);
DCL SPCPTR .CONTEXT DEF(CTRL) POS(17) INIT(CONTEXT);

DCL DD CONTEXT CHAR(96); /* PRIVATE CONTEXT FOR CIPHER INSTRUCTION */

/*****
...
INITIALIZE-CIPHER-CTRL:
  CPYBREP      CONTEXT, X'00'; /* resetting work-area */
  CALLX        .SEPT(GET-ENTRY), GET-OPERATION, *; /* primed read */

  CPYBLA       CIPHER-SOURCE, INBUF;
  CPYBLA       CTRL-SEQUENCE, X'01'; /* FIRST */
  CIPHER       .CIPHER-RESULT, CTRL, .CIPHER-SOURCE;

TRANSFORM-FILE:
  CPYBLA       CTRL-SEQUENCE, X'02'; /* MIDDLE */
  CIPHER       .CIPHER-RESULT, CTRL, .CIPHER-SOURCE;

  CALLX        .SEPT(GET-ENTRY), GET-OPERATION, *;
  CPYBLA       CIPHER-SOURCE, INBUF;
  B            TRANSFORM-FILE;

EOF-DETECTED:
  CPYBLA       CTRL-SEQUENCE, X'03'; /* FINAL */
  CIPHER       .CIPHER-RESULT, CTRL, .CIPHER-SOURCE;

  CPYBLAP      OUTBUF, IFCB-FILE, " "; /* filename in output */
  CVTHC        OUTBUF(33:40), CIPHER-RESULT; /* show hex values of result */
  CPYBLA       PARM-GENERAL, IFCB-FILE; /* also return as parameter */
  CPYBLA       PARM-GENERAL(11:40), OUTBUF(33:40);

  CALLX        .SEPT(PUT-ENTRY), PUT-OPERATION, *; /* write one record with result */
  CALLX        .SEPT(CLOSE-ENTRY), CLOSE-I, *;
  CALLX        .SEPT(CLOSE-ENTRY), CLOSE-O, *;
  RTX         *;
```

Here is a typical result:

```
TOPC                                21CDCDAC4E48FD5DA3676D5DD9B6A5DB2F3E0C8D
```

where we have decided to carry the hexadecimal representation of the hash value rather than its binary value.

## Clearing the File Before Output

In our Command Processing Program, CLFTRANS/QCLSRC, we deleted the file before executing the various file transformation programs. This was done because the default action when you open a file for output is to *add* records to the end of the file. Another reason was that having the delete operation outside of the file transformation program gives you more flexibility in the use of the program: sometimes you want to add records, sometimes you want to replace all the records with new records. But there are situations where you don't need or want this flexibility. So, how to, at open time, tell the system that you want to replace the records?

## Variable Length Part of the UFCB

You may recall from Chapter 12 that the User File Control Block, the UFCB, consists of a fixed part and a variable part. Each item of the variable part is headed by a 2-byte binary number identifying the type of the parameter item. We have already met the 'record-length' parameter, which has an identifier of **1**:

```
DCL DD UFCB-LENGTH-ID      BIN (2)  DEF(UFCB) POS(209) INIT(1);
DCL DD UFCB-RECORD-LENGTH  BIN (2)  DEF(UFCB) POS(211) INIT(132);
```

The 'clear member at open time yes/no' parameter has an identifier value of **8**, with the first bit in the following byte meaning "yes"; in other words:

```
DCL DD UFCB-CLEAR-ID      BIN (2)  DEF(UFCB) POS(213) INIT(8);
DCL DD UFCB-CLEAR-YES     CHAR(1)  DEF(UFCB) POS(215) INIT(X'80');
```

Remember to update the POS() values and the length of the UFCB itself.

## Blocking Records to Increase I/O Performance

You can increase I/O performance by reading several records ahead or by "batching-up" several records in each physical I/O "block". This is also controlled by an entry in the variable part of the UFCB:

```
DCL DD UFCB-BLOCK-RECORDS  BIN (2)  DEF(UFCB) POS(213) INIT(58);
DCL DD UFCB-BLOCK-YES     CHAR(1)  DEF(UFCB) POS(215) INIT(X'C0');
DCL DD UFCB-RECS-PER-BLOCK BIN (2)  DEF(UFCB) POS(216) INIT(20);
```

Remember to update the POS() values and the length of the UFCB itself.

## With/Without Level Check

It is possible to control whether file level checking should be done when the file is opened. The UFCB variable part entry is:

```
DCL DD UFCB-LEVEL-CHECK    BIN (2)  DEF(UFCB) POS(213) INIT(6);
DCL DD UFCB-LVLCHK-VALUE  CHAR(1)  DEF(UFCB) POS(215) INIT(X'xx');
```

Where **xx** should be 80 to include level check or 00 to exclude level check.

---

<sup>i</sup> **grok** /grok/, var. /grohk/ vt.

[from the novel "Stranger in a Strange Land", by Robert A. Heinlein, where it is a Martian word meaning literally 'to drink' and metaphorically 'to be one with'] The emphatic form is 'grok in fullness'. 1. To understand, usually in a global sense. Connotes intimate and exhaustive knowledge. Contrast [zen](#), which is similar supernal understanding experienced as a single brief flash. 2. Used of programs, may connote merely sufficient understanding. "Almost all C compilers grok the void type these days."

## Chapter 14

### Password Encryption on the AS/400

#### ***Password Protected Access***

The lowly password is still the main barrier to unauthorized access to the AS/400. In a more innocent age, the S/38 stored user names and passwords in the clear in an independent index object called the *Authorized User Table*: **QSYUPTBL** in library **QSYS**. The AS/400 still stores this information in the same place, but the password is no longer stored in the clear but is weakly encrypted. A fair amount of disinformation has been spread around about this issue. Below is a quote from News 3X/400 [August 1994, pg. 83] (well-known author's name withheld):

“It is virtually impossible to hack your way into an AS/400's password file. Even the most capable non-Rochester AS/400 programmers are thwarted in their efforts to find where the AS/400 stores passwords. And if someone does manage to uncover the physical location of the password file, the fun has just begun. Because OS/400 implements a scheme of double encryption for passwords (i.e. OS/400 encrypts the password with one key, encrypts the encryption with a different key, and then stores the result on disk), to find passwords, you would have to know not only both encryption keys but the encryption algorithm used).”

In this chapter we shall dispel some of the mystery and tell the sad truth about password encryption on the AS/400. All of this information is already public knowledge, so no deep secrets are being revealed. It is likely (one may be allowed to hope) that this chapter will soon become obsolete as IBM shores up password security on the iSeries/400.

#### ***Password Rules***

Passwords can be up to ten characters long and consist of the capital letters *A* through *Z*, the digits *0* through *9*, and the four characters pound (*#*), dollar (*\$*), underscore (*\_*), and the at sign (*@*). The first character cannot be a digit. There is a slight complication for code pages other than English because national language characters are mapped onto these four special characters in a rather arbitrary and *ad hoc* manner. Having a limited set of characters is a *severe* weakness, as the possible search space becomes much smaller. For the first character we have 30 possible values, for each of the rest we have 40 possible values, for a total of  $30 \cdot (40^{10} - 1) / (40 - 1) = 8.066 \cdot 10^{15}$  possible passwords. Large as this number is, it is still millions of times smaller than the full search space without restrictions on characters. As we shall see, the encryption is executed so poorly that a further factor of more than 64,000 is lost making brute-force decryption possible in a few hours. This is not entirely IBM's fault. Most of the blame for this should be placed on Microsoft.

There are many system values that control additional rules for forming a valid password. One of my favorites is “The **QPWDMAXLEN** system value controls the maximum number of characters in a password. This provides additional [*sic*] security by preventing users from specifying passwords that are too long”. [IBM SC41-8083-02, Application System/400, Security Reference, pg. 3-6]. Any of the other rules that prohibit certain character combinations or limit the use of certain constructs all reduce the search space and thus weakens the encryption.

#### ***Retrieving the Encrypted Password***

The **QSYRUPWD** API retrieves the encrypted password information for a given user profile. The API is entry 5723 in the SEPT (see appendix D). This API works with the Set Encrypted User Password (**QSYSUPWD**) API in that the APIs allow the administrator to more easily mirror the user profile activity on a second system based on the activity at the first system. The manual describes this information in a rather circumspect way as follows: The number of bytes of data available to be returned to the caller may increase from release to release but will always be a minimum of 2000 bytes. If the bytes available field is greater than the bytes returned field, the receiver variable cannot successfully be used as input to the **QSYSUPWD** API as not all encrypted password data will be returned by this API.

The **MIRTVPEW** program retrieves the encrypted password information and shows the hexadecimal representation of the information:

**44 1C26CBC1C4910CFCD71BC19F694F7FEF 3117672E0EC27808ADD5B1A41F2CB2C0**

We have split the information into three fields for reasons that will become clear shortly.

Here is the program. It is set up to retrieve my own encrypted password:

```
DCL SPCPTR .RECEIVER INIT(RECEIVER);
DCL DD RECEIVER CHAR(2000);
DCL DD RCV-BYTES-RETURNED BIN(4) DEF(RECEIVER) POS( 1);
DCL DD RCV-BYTES-AVAILABLE BIN(4) DEF(RECEIVER) POS( 5);
DCL DD RCV-USER-PROFILE CHAR(10) DEF(RECEIVER) POS( 9);
DCL DD RCV-DATA CHAR(100) DEF(RECEIVER) POS(19);

DCL SPCPTR .LENGTH INIT(LENGTH);
DCL DD LENGTH BIN(4) INIT(2000);

DCL SPCPTR .FORMAT INIT(FORMAT);
DCL DD FORMAT CHAR(8) INIT("UPWD0100");

DCL SPCPTR .USER INIT(USER);
DCL DD USER CHAR(10) INIT("LSVALGAARD");

DCL SPCPTR .ERROR INIT(ERROR);
DCL DD ERROR BIN(4) INIT(0);

DCL OL QSYRUPWD (.RECEIVER, .LENGTH, .FORMAT, .USER, .ERROR);

CALLX .SEPT(5723), QSYRUPWD, *;
CVTHC MSG-TEXT, RCV-DATA(1:35);
CALLI SHOW-MESSAGE, *, .SHOW-MESSAGE;
RTX *;
```

%INCLUDE SHOWMSG

We can compare the data returned with the contents of my entry in QSYUPTBL:

000000	D5D3E2E5	C1D3C7C1	C1D9C440	40404040	401C26CB	C1C4910C	FCD71BC1	9F694F7F	*NLSVALGAARD
000020	EF404040	40404040	40404000	4001F440	00000000	00000000	1B478474	93000800	*0
000040	00000000	00000000	81151A7B	51330000	44000031	17672E0E	C27808AD	D5B1A41F	* a
000060	2CB2C000	60404040	40404040	40404040	40404040	40404040	00000040	40404040	* { -
000080	40404040	40404040	40404040	40404040	40404040	40404040	40404040	40404040	* }

It is at once apparent that there are *two* 16-byte encrypted passwords. If I change my password, both of these values change as well. The x'44' is not part of the encrypted passwords, but represents possibly an indicator specifying which encryption methods are being used. In order not to give my password away, I've changed it momentarily to be equal to my user name. In V5R1 and later there are three encryptions of the password.

### Algorithm for the 1<sup>st</sup> Encrypted Password

The algorithm for computing the first encrypted password is given by Murphy, Rieth and Stevens of IBM Corporation. An explanation of this particular algorithm is freely available as an upgrade of RFC 1205 at <http://www.ietf.org/internet-drafts/draft-ietf-tn3270e-tn5250e-05.txt> The 05.txt suffix may change over time. It was 04.txt, and before that 03.txt...

We quote part of their description here:

[...] Client Access uses well-known [DES](#) encryption algorithms to create encrypted passwords. A client can generate compatible encrypted passwords if they follow these steps, details of which can be found in the Federal Information Processing Standards 46-2.

1. Padded\_PW = Left justified user password padded to the right with '40'X to 8 bytes. The users password must be left justified in an 8 byte variable and padded to the right with '40'X up to an 8 byte length. If the user's password is 8 bytes in length, no padding would occur. For computing password substitutes for passwords of length 9 and 10 see [...] below. If the password is not in EBCDIC, it must be converted to EBCDIC uppercase.

2. `XOR_PW = Padded_PW xor '5555555555555555'` The padded password is Exclusive OR'ed with 8 bytes of '55'. [this simply inverts the value of every second bit]

3. `SHIFT_RESULT = XOR_PW << 1`. The entire 8-byte result is shifted 1 bit to the left; the leftmost bit value is discarded, and the rightmost bit value is cleared to 0.

4. `PW_TOKEN = DES_ECB_mode(SHIFT_RESULT, /*key*/ userID_in_EBCDIC_uppercase /*data*/ )` This shifted result is used as key to the Data Encryption Standard [...] to encipher the user identifier. When the user identifier is less than 8 bytes, it is left justified in an 8 byte variable and padded to the right with '40'. When the user identifier is 9 or 10 bytes, it is first padded to the right with '40' to a length of 10 bytes. Then bytes 9 and 10 are "folded" into bytes 1-8 using the following algorithm:

Bit 0 is the high-order bit (i.e. has value of '80').

Byte 1, bits 0 and 1 are replaced with byte 1, bits 0 and 1 Exclusive OR'ed with byte 9, bits 0 and 1.

Byte 2, bits 0 and 1 are replaced with byte 2, bits 0 and 1 Exclusive OR'ed with byte 9, bits 2 and 3.

Byte 3, bits 0 and 1 are replaced with byte 3, bits 0 and 1 Exclusive OR'ed with byte 9, bits 4 and 5.

Byte 4, bits 0 and 1 are replaced with byte 4, bits 0 and 1 Exclusive OR'ed with byte 9, bits 6 and 7.

Byte 5, bits 0 and 1 are replaced with byte 5, bits 0 and 1 Exclusive OR'ed with byte 10, bits 0 and 1.

Byte 6, bits 0 and 1 are replaced with byte 6, bits 0 and 1 Exclusive OR'ed with byte 10, bits 2 and 3.

Byte 7, bits 0 and 1 are replaced with byte 7, bits 0 and 1 Exclusive OR'ed with byte 10, bits 4 and 5.

Byte 8, bits 0 and 1 are replaced with byte 8, bits 0 and 1 Exclusive OR'ed with byte 10, bits 6 and 7.

Handling passwords of length 9 and 10:

1. Generate `PW_TOKENa` by using characters 1 to 8 of the password and steps 1-4 from the previous section.

2. Generate `PW_TOKENb` by using characters 9 and 10 and steps 1-4 from the previous section. In this case `Padded_PW` from step 1 will be characters 9 and 10 padded to the right with '40', for a total length of 8.

Some additional steps having to do with sequencing of transmission blocks have been omitted as not being applicable to just computing the encrypted password itself. Note that if the password is less than 9 characters long, only one 8-byte `PW_TOKEN` is computed and the remainder of the 16-byte encrypted password is left blank. For longer passwords `PW_TOKENa` and `PW_TOKENb` are concatenated to form the final 16-byte encrypted password.

The curious step of shifting the password one bit to the left comes from the fact that just using the original password as the key is actually not a good idea, because the DES algorithm does not use the *low-order* bit of each character. This is the infamous weakening of the algorithm from 64 bits to 56 bits as demanded by the NSA for reasons never convincingly explained. Since DES does not use the low-order bits, we would be throwing away information if we were using the password *as-is*. Now, because of the structure of the EBCDIC character set, one can actually omit the *high-order* bit of every valid password character without losing information. So, an obvious solution is simply to shift the password to the left one bit. This preserves all information and gives DES a low-order bit to throw away.

Let's write MI-code to accomplish each step:

```
CPYBLAP      KEY, PWD, X' 40' ;          /* step 1 */
XOR(S)       KEY, X' 5555555555555555' ; /* step 2 */
```

The "Exclusive Or"-instruction, **XOR**, performs a bit-wise Boolean *exclusive or* operation where two bits transform to a bit value of 1 if they are different, otherwise to a bit value of 0.

```
ADDLC(S)     KEY, KEY;                   /* step 3 */
```

The "Add Logical Character"-instruction, **ADDLC**, is a convenient way of shifting a character string one bit to the left (adding it to itself is the same as multiplying by two, which is the same as shifting one bit to the left).

Alternatively, we could have used the "Copy Bits with Left Logical Shift"-instruction, **CPYBTLLS**, which copies the bit string value of the *source* operand to the bit string defined by the *receiver* operand with a left

logical shift of the source controlled by the value of the *shift* (the 3<sup>rd</sup>) operand. The shift operand must be an unsigned BIN(2) number, an unsigned immediate value, or a character CHAR(2) value containing the shift count:

```
CPYBTLLS    KEY, KEY, 1;                /* step 3, alternate */
```

Folding the excess characters of the user name (if necessary) proceeds by shifting the characters into position, extracting the topmost two bits (bit 0 and 1) and xor'ing them into the user name:

```
CMPBLA(B)   USER(9:2), " " /EQ(NEXT);

CPYBTLLS    WORK, USER(9:2), 0;          /* step 4, folding */
AND(S)      WORK, X'CO';                /* extract bits 0 and 1 */
XOR(S)      USER(1:1), WORK;            /* XOR into USER */

CPYBTLLS    WORK, USER(9:2), 2;
AND(S)      WORK, X'CO';
XOR(S)      USER(2:1), WORK;

CPYBTLLS    WORK, USER(9:2), 4;
AND(S)      WORK, X'CO';
XOR(S)      USER(3:1), WORK;

CPYBTLLS    WORK, USER(9:2), 6;
AND(S)      WORK, X'CO';
XOR(S)      USER(4:1), WORK;

CPYBTLLS    WORK, USER(9:2), 8;
AND(S)      WORK, X'CO';
XOR(S)      USER(5:1), WORK;

CPYBTLLS    WORK, USER(9:2), 10;          /* note the numerical */
AND(S)      WORK, X'CO';                /* relationship between */
XOR(S)      USER(6:1), WORK;            /* the various numbers: */

CPYBTLLS    WORK, USER(9:2), 12;          /* 12 = (7-1)*2 */
AND(S)      WORK, X'CO';
XOR(S)      USER(7:1), WORK;

CPYBTLLS    WORK, USER(9:2), 14;          /* 14 = (8-1)*2 */
AND(S)      WORK, X'CO';
XOR(S)      USER(8:1), WORK;            /* end of step 4 */
```

These eight pieces of almost identical code cry out for a loop, where the shift amount can be computed from the character position:

```
CPYNV       N, 8;                        /* start from the end */
FOLD:
SUBN        SHIFT, N, 1;
ADDN(S)     SHIFT, SHIFT;                /* shift amount = (N-1)*2 */
CPYBTLLS    WORK, USER(9:2), SHIFT;
AND(S)      WORK, X'CO';
XOR(S)      USER(N), WORK;
SUBN(SB)    N, 1/POS(FOLD);
```

## Using CIPHER to DES-Encrypt

The CIPHER-instruction, we used in chapter 13 for calculating a hash value, can also be used to execute the DES encryption algorithm. This is what CIPHER was originally designed to do. Because of the silly U.S. laws regarding encryption, CIPHER was deliberately crippled in the AS/400 to the point where about the only thing it can be used for is encryption of passwords. *E.g.* the maximum amount of data that can be encrypted was reduced to 64 bytes, and no decryption is available. The control block is slightly different for DES encryption than for the hash calculation:

```
DCL DD CONTROL CHAR(32);
DCL DD CTRL-FUNCTION CHAR(2) DEF(CONTROL) POS(1) INIT(X'0002'); /* DES encrypt */
DCL DD CTRL-SIZE BIN(2) DEF(CONTROL) POS(3) INIT(8); /* bytes to encrypt */
DCL DD CTRL-OPTION CHAR(1) DEF(CONTROL) POS(5) INIT(X'00'); /* simple encryption */
DCL DD KEY CHAR(8) DEF(CONTROL) POS(6); /* encryption key */

DCL SPCPTR .CRYPT INIT(CRYPT); DCL DD CRYPT CHAR(16); /* Encryption result */
DCL SPCPTR .USER INIT(USER); DCL DD USER CHAR(10); /* USER name */

CIPHER      .CRYPT, CONTROL, .USER;
```



The **MI ENCPWD** program pulls it all together. Note how we have two loops, using a *sentinel* at the end of the password, which we made 16+8 characters long internally:

```
DCL DD EPWD CHAR(16); /* ENCRYPTED PASSWORD */
DCL DD PWD CHAR(24); /* GIVEN PASSWORD TO 24 POSITIONS */
DCL DD WORK CHAR(1);
DCL DD SHIFT BIN(2) UNSGND;
DCL DD N BIN(2);
DCL DD P BIN(2);

DCL DD CONTROL CHAR(32);
DCL DD CTRL-FUNCTION CHAR(2) DEF(CONTROL) POS(1) INIT(X'0002');
DCL DD CTRL-SIZE BIN(2) DEF(CONTROL) POS(3) INIT(8);
DCL DD CTRL-OPTION CHAR(1) DEF(CONTROL) POS(5) INIT(X'00');
DCL DD KEY CHAR(8) DEF(CONTROL) POS(6);

DCL SPCPTR .USER INIT(USER);
DCL DD USER CHAR(10);

DCL SPCPTR .CRYPT INIT(CRYPT);
DCL DD CRYPT CHAR(8);

SET-USER-PWD:
CPYBLAP USER, "LSVALGAARD", " ";
CPYBLAP PWD, "LSVALGAARD", " "; /* also clears sentinel */

CPYBREP EPWD, " "; /* CLEAR ENCRYPTED PASSWORD */
CMPBLA(B) USER(9:2), " /EQ(PREPARE-TO-ENCRYPT);

CPYNV N, 8; /* START FROM THE END */
FOLD:
SUBN SHIFT, N, 1;
ADDN(S) SHIFT, SHIFT;
CPYBTLIS WORK, USER(9:2), SHIFT;
AND(S) WORK, X'CO'; /* TOP TWO BITS */
XOR(S) USER(N:1), WORK;
SUBN(SB) N, 1/POS(FOLD);

PREPARE-TO-ENCRYPT:
CPYNV P, 1;
ENCRYPT:
CPYBLAP KEY, PWD(P:8), X'40'; /* STEP 1 */
CMPBLA(B) KEY, X'40' /EQ(SHOW); /* compares 1st character only */

XOR(S) KEY, X'5555555555555555'; /* STEP 2 */
ADDLC(S) KEY, KEY; /* STEP 3 */
CIPHER .CRYPT, CONTROL, .USER;
CPYBLA EPWD(P:8), CRYPT;
ADDN(SB) P, 8/POS(ENCRYPT);

SHOW:
CVTHC MSG-TEXT(1:32), EPWD;
CALLI SHOW-MESSAGE, *, .SHOW-MESSAGE;
RTX *;

%INCLUDE SHOWMSG
```

The result is precisely the first encrypted password that we retrieved from the user table:

```
Type reply (if required), press Enter.
From . . . : LSVALGAARD 10/01/00 19:16:21
1C26CBC1C4910CFCD71BC19F694F7FEF
```

### Algorithm for the 2<sup>nd</sup> Encrypted Password

In order that people should be able to logon to the AS/400 directly from Windows, the AS/400 also supports the LANMAN password encryption by storing the LANMAN encrypted password as the 2<sup>nd</sup> encrypted value in the user table. The algorithm for computing the LANMAN encrypted password has been all over the Internet for several years. A C++ implementation can be downloaded from:

<http://download.sourceforge.net/mirrors/NetBSD/NetBSD-current/pkgsrc/net/ppp-mppe/patches/patch-bj>

LANMAN is Microsoft's Manager for Local Area Networks. There are some fascinating stories out there about LANMAN security. One of the better references is <http://www.l0pht.com/l0phtcrack/rant.html>. We quote:



I didn't ask to be cc'd into the rantings of the MS Borg Marketing Juggernaut but since I'm here... I find this hilarious. The people at MS should know better. I haven't been following this thread tremendously but I've seen bits and pieces. Recently there was an atrocious article in WindowsNT magazine, where they stated it would take 5000 or so years to break the passwords; thus put policy in place to have users change their passwords every 2500 years. HELLO? I think these people aren't getting it. Let's shed some light on things shall we?

1. Thank you very little MS for dropping any reference to the l0pht, hobbit, or myself in reference to your recent LM-Hash fix. If this is how you "correspond" with people who point out problems to you it's no wonder that people prefer to release things to the public instead of your "proper" channels.
2. MS agrees that the LM hash is a horrible implementation from a security standpoint. They respond with "well we didn't write the protocol that was IBM". [My comment: IBM blames MS]
3. When MS had the chance to do things a different way (*i.e.* Network challenge/response obfuscation on NT boxes) they implemented it based upon LM techniques to break up components.
5. MS can't swallow their pride enough to say "oops", even in technical circles where they don't have to worry about the general public misinterpreting things.
6. For the LM hash you only have to break 7 characters, not 14!
7. MS keeps talking about the NT hash being so secure while refusing to talk about how weak the LM hash is. Guess what, you probably won't be able to use the "added security" of the NT hash on your network. Why keep talking about something people can't use?
9. [...] Now, let's rip apart why it is so trivial to go through the LM hash on the network. [...] We already know that you only have to go through 7 characters to retrieve passwords (up to 14 chars in length) in the LM hash, and that since there is no salting being done, constants show up all over the place giving away too much information and speeding up attacks tremendously. The 1<sup>st</sup> 8 bytes are derived from the first seven characters of the password and the 2<sup>nd</sup> 8 bytes are derived from the 8<sup>th</sup> through 14<sup>th</sup> characters of the password. If the password is less than 8 characters then the second half will always be: 0xAAD3B435B51404EE. [...] The 7 byte strings are str\_to\_key'd (if you will) into 8 byte odd parity DES keys. [...] The LM hash is incredibly weak and your more secure NT hash is brought down to the lowest common denominator.

Interchange MS with IBM, NT with OS/400, and the story rings just as true. What is particular interesting is that both parties blame each other for the weakness of the algorithm.

Here is an excerpt from "Virtual Interview with Jeremy Allison and Andrew Tridgell" which you can find at <http://www2.linuxjournal.com/lj-issues/issue50/2900.html>:

**John:** *So the protocol is just whatever they could get to work?*

**Jeremy:** Yeah, basically. The protocol is really horrible. It sort of grew like a wart. You can tell by looking at it. So, what happened was once we got the long file name support working, the next thing we wanted to get working was encrypted password support. We hated having to put plaintext passwords out. There was a "magic constant" that Microsoft actually refused to give out to anybody. It's used in the DES encryption for the LanMan password support. They refused to give it to us without an NDA (Non-Disclosure Agreement). We replied "We're not giving you an NDA, because we're publishing source."

There it sat for a while until I went to a Microsoft NT conference up in Seattle and had a conversation with a, uhh, sort of mentally challenged Microsoft person who was basically saying "Why do you want to break open all our LanMan systems?", and I said "I don't, I want to interoperate with them," which I suppose to her was the same thing. So, she put me in touch with a Microsoft programmer, Richard Ward, who is in charge of some of their security stuff. I chatted with him, and he couldn't tell me what it was, but he said, if you forward me a message I'll put you in touch with somebody who knows more about it.

Now what happened - and this is really interesting - the programmer he put me in touch with basically said, “Well, what do you need to know?” I thought, well, if I ask him the question directly, I won’t get the answer. He’ll say, “I can’t tell you that.” So, I phrased the question differently. The LanMan encryption is a two stage operation, and we needed the middle bit. I phrased the question in such a way that we could work backwards from what he told us. I asked, if we encrypt this password, what will the second stage be? Once he told us, it was trivial to reverse it and out popped the magic constant. And once I had the constant, I actually took a look at a hexdump of the Microsoft redirector, and the magic constant is contained in a field of zeros as plain as day.

**Andrew:** It actually took us a day or two to realize that the information he sent gave us what we needed. Jeremy and I both realized it independently within a few hours of each other. It was an unexpected bonus!

**John:** *So it was easy to see the magic constant once you knew where to look? [Note: the magic constant is KGS!@#\$.]*

A simple way of finding the magic constant is to use a known password as a key and *decrypt* the encrypted value giving the plain text, i.e. the magic constant.

## Why Seven Characters Instead of Eight?

The first seven characters of the 10-character password go into constructing the first piece of the encrypted password string. But why *seven*? The key used in DES is 64 bits long (8 characters). However, of these, DES only uses 56 bits, as the low-order bit of each character is ignored. In an unconvincing attempt to hide the fact that DES was deliberately crippled by reducing the key length from 64 bits to 56 bits, the low-order bit was called “the parity bit”. Just dropping the low-order bit is undesirable as a leading B (X‘C2’) would encrypt the same as a C (X‘C3’). To build the key without losing information we proceed as follows:

Treat the password as one long string of bits. Take the first 7 bits and place them left-justified in the first character position of a key work area. Then take the *next* 7 bits of the password and place *them* left-justified in the *second* character position of the key work area, and so on, until all eight character positions of the key work area have been filled (this is also when all seven characters of the password have been used). The low-order bit of each key character can be set to anything you like (or left *as-is*) since it is not used by the DES algorithm anyway.

The following MI-instructions build the **KEY**:

```
CPYBTLLS KEY(1:1), PWD, 0; /* copy bits TO, FROM, SHIFT */
CPYBTLLS KEY(2:1), PWD, 7;
CPYBTLLS KEY(3:1), PWD, 14;
CPYBTLLS KEY(4:1), PWD, 21;
CPYBTLLS KEY(5:1), PWD, 28; /*      SHIFT = (N-1)*7      */
CPYBTLLS KEY(6:1), PWD, 35;
CPYBTLLS KEY(7:1), PWD, 42;
CPYBTLLS KEY(8:1), PWD, 48;
```

A loop seems in order, so:

```
CPYNV      N, 8; /* START FROM THE END */
BUI LD:
SUBN      SHIF T, N , 1;
MULT(S)   SHIF T, 7;
CPYBTLLS  KEY(N:1), PWD, SHIF T;
SUBN(SB)  N, 1/POS(BUI LD);
```

## The Magic Constant

The *Magic Constant* “KGS!@#\$.” is used as clear-text in a DES-encryption with the key we’ve just built. The last five characters are what you get if you hold down the shift key on your keyboard and press the keys marked 1, 2, 3, 4, and 5 on the row above QWERTY, namely “!@#\$.”. The first three characters are “KGS”, obviously somebody’s initials. In short, the secret clear-text to encrypt with the password is the ASCII representation of “KGS!@#\$.” and the padding character is the null character (same in EBCDIC and ASCII). We can translate from EBCDIC to ASCII the same way we did in chapter 13:

```

DCL DD ASCII-TABLE CHAR(256);
DCL DD *(16) CHAR(16) DEF(ASCII-TABLE) POS(1) INIT
(X' 00000000000000000000000000000000', /* 00-0F ----- */
X' 00000000000000000000000000000000', /* 10-1F ----- */
X' 00000000000000000000000000000000', /* 20-2F ----- */
X' 00000000000000000000000000000000', /* 30-3F ----- */
X' 00000000000000000000000002E2C282B00', /* 40-4F ----- <(+ */
X' 260000000000000000000000021242A293B00', /* 50-5F &-----!$*);- */
X' 2D2F0000000000000000000002C255F3E3F', /* 60-6F -/-----,%_>? */
X' 00000000000000000000000003A2340273D22', /* 70-7F -----:#@'=" */
X' 0041424344454647484900000000000000', /* 80-8F -abcdefghi ----- */
X' 004A4B4C4D4E4F50515200000000000000', /* 90-9F -jklmnopqr----- */
X' 0000535455565758595A00000000000000', /* A0-AF --stuvwxyz----- */
X' 00000000000000000000000000000000', /* B0-BF ----- */
X' 0041424344454647484900000000000000', /* C0-CF -ABCDEFGH I ----- */
X' 004A4B4C4D4E4F50515200000000000000', /* D0-DF -JKLMNOPOR----- */
X' 0000535455565758595A00000000000000', /* E0-EF --STUVWXYZ----- */
X' 3031323334353637383900000000000000' ); /* F0-FF 0123456789----- */
/* NOTE THAT 'BLANK' TRANSLATES TO '00' */

```

```

TRANSLATE:
XLATEWT USER, "KGS!@#$$%", ASCII-TABLE;
XLATEWT PWD, PWD, ASCII-TABLE;

CPYNV N, 8; /* START FROM THE END */
BUILD:
SUBN SHIF T, N, 1;
MULT(S) SHIF T, 7;
CPYBTLLS KEY(N:1), PWD, SHIF T;
SUBN(SB) N, 1/POS(BUILD);
CIPHER .CRYPT, CONTROL, .USER;

```

Because we need to encipher both 7-byte pieces of the password, we add an extra loop:

```

CPYNV P, 1; /* character position in password */
CPYNV E, 1; /* character position in encrypted password */
NEXT:
CPYNV N, 8; /* character position in DES key */
BUILD:
SUBN SHIF T, N, 1;
MULT(S) SHIF T, 7;
CPYBTLLS KEY(N:1), PWD(P:8), SHIF T;
SUBN(SB) N, 1/POS(BUILD);
CIPHER .CRYPT, CONTROL, .USER;
CPYBLA EPWD(E:8), CRYPT;

ADDN(S) P, 7;
ADDN(S) E, 8;
CMPNV(B) P, 14/LO(NEXT);

```

The result matches the second encrypted password:

```

Type reply (if required), press Enter.
From . . . : LSVALGAARD 10/01/00 22:44:22
1C26CBC1C4910CFCD71BC19F694F7FEF 3117672E0EC27808ADD5B1A41F2CB2C0

```

## The Compleat *MI ENCPWD* Program

The program to calculate both encrypted passwords is now ready:

```

DCL DD EPWD CHAR(16); /* ENCRYPTED PASSWORD */
DCL DD PWD CHAR(24); /* GIVEN PASSWORD TO 24 POSITIONS */
DCL DD WORK CHAR(1);
DCL DD SHIF T BIN(2) UNSGND;
DCL DD N BIN(2);
DCL DD P BIN(2);
DCL DD E BIN(2);

DCL DD ASCII-TABLE CHAR(256);
DCL DD *(16) CHAR(16) DEF(ASCII-TABLE) POS(1) INIT
(X' 00000000000000000000000000000000', /* 00-0F ----- */
X' 00000000000000000000000000000000', /* 10-1F ----- */
X' 00000000000000000000000000000000', /* 20-2F ----- */
X' 00000000000000000000000000000000', /* 30-3F ----- */
X' 00000000000000000000000002E2C282B00', /* 40-4F ----- <(+ */
X' 260000000000000000000000021242A293B00', /* 50-5F &-----!$*);- */
X' 2D2F0000000000000000000002C255F3E3F', /* 60-6F -/-----,%_>? */
X' 00000000000000000000000003A2340273D22', /* 70-7F -----:#@'=" */
X' 0041424344454647484900000000000000', /* 80-8F -ABCDEFGH I ----- */

```

```

X' 004A4B4C4D4E4F505152000000000000', /* 90-9F -JKLMNOPQR----- */
X' 0000535455565758595A000000000000', /* A0-AF --STUVWXYZ----- */
X' 00000000000000000000000000000000', /* B0-BF ----- */
X' 00414243444546474849000000000000', /* C0-CF -ABCDEFGH----- */
X' 004A4B4C4D4E4F505152000000000000', /* D0-DF -JKLMNOPQR----- */
X' 0000535455565758595A000000000000', /* E0-EF --STUVWXYZ----- */
X' 30313233343536373839000000000000'); /* F0-FF 0123456789----- */

DCL DD CONTROL CHAR(32);
DCL DD CTRL-FUNCTION CHAR(2) DEF(CONTROL) POS(1) INIT(X' 0002' );
DCL DD CTRL-SIZE BIN(2) DEF(CONTROL) POS(3) INIT(8);
DCL DD CTRL-OPTION CHAR(1) DEF(CONTROL) POS(5) INIT(X' 00' );
DCL DD KEY CHAR(8) DEF(CONTROL) POS(6);

DCL SPCPTR .USER INIT(USER);
DCL DD USER CHAR(10);

DCL SPCPTR .CRYPT INIT(CRYPT);
DCL DD CRYPT CHAR(8);

SET-USER-PWD:
CPYBLAP USER, "LSVALGAARD", " ";
CPYBLAP PWD, "LSVALGAARD", " ";

FIRST-ENCRYPTED-PASSWORD:
CPYBREP EPWD, " "; /* CLEAR ENCRYPTED PASSWORD */
CMPBLA(B) USER(9:2), " " /EQ(PREPARE-TO-ENCRYPT);
CPYNV N, 8; /* START FROM THE END */
FOLD:
SUBN SHIF, N, 1;
ADDN(S) SHIF, SHIF;
CPYBTLLS WORK, USER(9:2), SHIF;
AND(S) WORK, X' C0' ; /* TOP TWO BITS */
XOR(S) USER(N:1), WORK;
SUBN(SB) N, 1/POS(FOLD);

PREPARE-TO-ENCRYPT:
CPYNV P, 1;
ENCRYPT:
CPYBLAP KEY, PWD(P:8), X' 40' ; /* STEP 1 */
CMPBLA(B) KEY, X' 40' /EQ(SHOW);

XOR(S) KEY, X' 5555555555555555' ; /* STEP 2 */
ADDN(S) KEY, KEY; /* STEP 3 */
CIPHER .CRYPT, CONTROL, .USER;
CPYBLA EPWD(P:8), CRYPT;
ADDN(SB) P, 8/POS(ENCRYPT);

SHOW:
CPYBREP MSG-TEXT, " ";
CVTHC MSG-TEXT(1:32), EPWD; /* 1ST ENCRYPTED PASSWORD */

SECOND-ENCRYPTED-PASSWORD:
XLATEWT USER, "KGS!@#%$", ASCII-TABLE;
XLATEWT PWD, PWD, ASCII-TABLE;

CPYNV P, 1; /* CHARACTER POSITION IN PASSWORD */
CPYNV E, 1; /* CHARACTER POSITION IN ENCRYPTED PASSWORD */
NEXT:
CPYNV N, 8; /* CHARACTER POSITION IN DES KEY */
BUILD:
SUBN SHIF, N, 1;
MULT(S) SHIF, 7;
CPYBTLLS KEY(N:1), PWD(P:8), SHIF;
SUBN(SB) N, 1/POS(BUILD);
CIPHER .CRYPT, CONTROL, .USER;
CPYBLA EPWD(E:8), CRYPT;

ADDN(S) P, 7;
ADDN(S) E, 8;
CMPNV(B) P, 14/LO(NEXT);

CVTHC MSG-TEXT(34:32), EPWD; /* 2ND ENCRYPTED PASSWORD */
CALLI SHOW-MESSAGE, *, .SHOW-MESSAGE;
RTX *;

```

%INCLUDE SHOWMSG

I hope you appreciate how elegantly these things can be written in MI. Try to look at some of C++ implementations.

You can now see why the OS/400 password encryption is so poor, treating each part of the password independently weakens both encryption methods. For the first method, passwords longer than eight characters are not more secure than passwords eight characters long. For the second method, passwords longer than seven characters are not more secure than passwords seven characters long.

## ***The Brute-Force Attack***

Because only passwords up to seven characters need be considered, the total number of possible passwords to try is only  $30 \cdot (40^0 + 40^1 + 40^2 + 40^3 + 40^4 + 40^5 + 40^6) = 30 \cdot (40^7 - 1) / (40 - 1) = 126,030,769,230$ . Although this number 10 years ago looked forbidding, it is not so today. On a 700 MHz Pentium III PC with MMX instructions it is possible to search more than 7,000,000 keys per second. At this speed, *any* password can be cracked in less than six hours. NEWS/400 recently [published](http://inet.uni2.dk/~svolaf) a reference to a program written by Svend Olaf Mikkelsen. The program can be downloaded freely from his home page at <http://inet.uni2.dk/~svolaf>. The program can be started with parameters to search a smaller slice of the search space, e.g. passwords starting with A through F. In this way, a parallel attack can be carried out using several PCs. With six PCs running in parallel, the password of any user, including **QSECOFR**, can be cracked in less than an hour, even if the password were carefully chosen to be difficult to guess (such as 'D@1X7W#').

## ***Passwords Stored in the Clear!***

Some people may say; "...but you need to have special \*SECADM authority to use the **QSYRUPWD** and the **QSYRUPWD** API's and you need \*ALLOBJ authority to get near the **QSYUPTBL** object...". But what does the strength of the password encryption and the difficulty of accessing the user table matter, if passwords were stored in the clear somewhere else on the system? Incredibly, this was indeed the case. Many people have known for a decade that there were flaws in OS/400's treatment of passwords. A fundamental rule when dealing with sensitive information is that all storage where this resides be cleared immediately after use. This did not happen on the AS/400. Not only in one place, but in *two* places, was the *clear text* copy of the password left intact. Only when this was widely publicized recently did IBM take action and issued two PTFs to remedy the situation (the second PTF was even defective). Although the OS/400 security team at IBM should be commended on taking action in spite of being low on the totem pole when it comes to allocation of resources, the whole story (weakened algorithms, reduced key space, passwords in clear) is really inexcusable. It is clear that IBM (and Microsoft) could benefit from some peer review. It is also clear that this is not going to happen any time soon.

## ***A Password Checking Utility***

We can put the techniques learnt in this chapter to practical use. Let's write a program to check if a given user profile/password combination is valid. This is simply a matter of combining code from the other programs in the present chapter. Call the resulting program, **MI CHKPWD**, with three parameters, the first one being the name of the user profile, the second being the password to check, and the third is a return value: "Y" if the password is correct, and "N" otherwise. Here is the program:

```
DCL SPCPTR .P1 PARM;
DCL DD PARM-USER      CHAR(10) BAS(. P1);

DCL SPCPTR .P2 PARM;
DCL DD PARM-PASSWORD CHAR(10) BAS(. P2);

DCL SPCPTR .P3 PARM;
DCL DD PARM-RESULT    CHAR(1)  BAS(. P3); /* Y = OK, N = NOT OK */

DCL OL PARMS(. P1, . P2, . P3) EXT PARM MIN(3);

DCL SPCPTR .RECEIVER INIT(RECEIVER);
DCL DD RECEIVER CHAR(2000);
DCL DD RCV-BYTES-RETURNED BIN(4) DEF(RECEIVER) POS( 1);
DCL DD RCV-BYTES-AVAILABLE BIN(4) DEF(RECEIVER) POS( 5);
DCL DD RCV-USER-PROFILE CHAR(10) DEF(RECEIVER) POS( 9);
DCL DD RCV-DATA          CHAR(1982) DEF(RECEIVER) POS(19);
DCL DD RCV-ENCRYPTED-PASSWORD CHAR(16) DEF(RCV-DATA) POS(2);

DCL SPCPTR .LENGTH INIT(LENGTH);
DCL DD LENGTH BIN(4) INIT(2000);
```

```

DCL SPCPTR .FORMAT INIT(FORMAT);
DCL DD      FORMAT CHAR(8) INIT("UPWDO100");

DCL SPCPTR .USER   INIT(USER);
DCL DD      USER   CHAR(10);

DCL SPCPTR .ERROR  INIT(ERROR);
DCL DD      ERROR  BIN(4) INIT(0);

DCL OL QSYRUPWD (.RECEIVER, .LENGTH, .FORMAT, .USER, .ERROR);

DCL SYSPTR .SEPT(6440) BAS(SEPT-POINTER);
DCL SPC PROCESS-COMMUNICATION-OBJECT BASPCO;
DCL SPCPTR SEPT-POINTER DIR;

DCL DD EPWD CHAR(16); /* ENCRYPTED PASSWORD */
DCL DD PWD CHAR(24); /* GIVEN PASSWORD TO 24 POSITIONS */
DCL DD WORK CHAR(1);
DCL DD SHIFT BIN(2) UNSGND;
DCL DD N BIN(2);
DCL DD P BIN(2);

DCL DD CONTROL CHAR(32);
DCL DD CTRL-FUNCTION CHAR(2) DEF(CONTROL) POS(1) INIT(X'0002');
DCL DD CTRL-SIZE BIN(2) DEF(CONTROL) POS(3) INIT(8);
DCL DD CTRL-OPTION CHAR(1) DEF(CONTROL) POS(5) INIT(X'00');
DCL DD KEY CHAR(8) DEF(CONTROL) POS(6);

DCL SPCPTR .CRYPT INIT(CRYPT);
DCL DD      CRYPT CHAR(8);

/*****

ENTRY * (PARMS) EXT;
  CPYBLA      PARM-RESULT, "N"; /* ASSUME BAD */
  CPYBLAP     PWD, PARM-PASSWORD, " ";
  CPYBLA      USER, PARM-USER;
  CALLX      .SEPT(5723), QSYRUPWD, *;

  CPYBREP     EPWD, " "; /* CLEAR ENCRYPTED PASSWORD */
  CMPBLA(B)   USER(9:2), " " /EQ(PREPARE-TO-ENCRYPT);
  CPYNV       N, 8; /* START FROM THE END */

FOLD:
  SUBN        SHIFT, N, 1;
  ADDN(S)     SHIFT, SHIFT;
  CPYBTLLS    WORK, USER(9:2), SHIFT;
  AND(S)      WORK, X'CO'; /* TOP TWO BITS */
  XOR(S)      USER(N:1), WORK;
  SUBN(SB)    N, 1/POS(FOLD);

PREPARE-TO-ENCRYPT:
  CPYNV       P, 1;
ENCRYPT:
  CPYBLAP     KEY, PWD(P:8), X'40'; /* STEP 1 */
  CMPBLA(B)   KEY, X'40' /EQ(CHECK);

  XOR(S)      KEY, X'5555555555555555'; /* STEP 2 */
  ADDLC(S)    KEY, KEY; /* STEP 3 */
  CIPHER      .CRYPT, CONTROL, .USER;
  CPYBLA      EPWD(P:8), CRYPT;
  ADDN(SB)    P, 8/POS(ENCRYPT);

CHECK:
  CMPBLA(B)   EPWD, RCV-ENCRYPTED-PASSWORD /NEQ(=+2);
  CPYBLA      PARM-RESULT, "Y";
  BRK "1";
  RTX        *;

```

To run the program you must, of course, have authority to retrieve the encrypted password.

### ***DMPSYSOBJ in V4R5 and Later***

In a silly attempt to “plaster over” the exposure described in this chapter, IBM has “doctored” the **DMPSYSOBJ** command to show the encrypted passwords as blanks, that is to “lie” about what is in the **QSYUPTBL** object. This, of course, fools nobody. The data is still there, as SST (or the Memory Explorer developed in chapter 26) will readily show.

### ***Real Security***

We’ll close this chapter with these words from Bruce Schneier (Preface to [Applied Cryptography](#)):

“If I take a letter, lock it in a safe, hide the safe somewhere in New York, then tell you to read the letter, that’s not security. That’s obscurity. On the other hand, if I take a letter and lock it in a safe, and then give you the safe along with the design specifications of the safe and a hundred identical safes with their combinations so that you and the world’s best safecrackers can study the locking mechanism - and you still can’t open the safe and read the letter - that’s security.”

## Chapter 15

### Program Validation Value

#### ***The Program Validation Value***

During an AS/400 program encapsulation operation, a built-in OS/400 system routine calculates a Program Validation Value that takes into account the object domain, the program state, and the instruction stream. This process imprints each program with a binary value that may be verified at any time to ensure that the program has not been altered since its creation. One of the alterations that is much sought after is to change the program's state from user state to system state. System state programs are privileged and can access sensitive system areas.

#### **States and Domains**

The Program Validation Value (PVV) was introduced with V1R3 to improve security. The *state* attribute is given only to executable objects such as programs. The domain attribute is given to all objects that are visible to the user. All system state programs may use any object on the system for which the user has the necessary authority. User state programs, however, may only reference user domain objects. All user-created programs are assigned user state and user domain attributes. Operating system programs are mostly assigned system state/system domain attributes. Some that are meant to be called by user programs (APIs) are assigned either system state/user domain or inherit state/user domain attributes as shown in appendix 4.

Almost all user-created non-executable objects, such as database files, commands, and job descriptions, are placed in the system domain. This means that objects created by an application programmer can only be accessed through the IBM-supplied operating system APIs. It also means that no user-created programs are able to activate or call operating system programs that are not meant for interaction with user-created programs. Furthermore, it is possible for the system to check if a user-state program tries to execute so-called *blocked* MI-instructions that are reserved for system-state programs. It does that by checking bit 56 in the Machine State Register (MSR) as we discussed in chapter 11.

#### **Saving/Restoring Programs**

The PVV is stored with the program when you *Save* the program object. If you *had* altered the program (e.g. using SST on a system where you are authorized to use SST), the idea is that the PVV would now be incorrect. The *Restore* process on a different system (where you have less authority) could check the PVV and either refuse to restore the program to your user profile, or, at least reset, the program to user state, if the PVV is incorrect. The assumption made by IBM here is that the PVV is cryptographically sound in the sense that it is infeasible to change the program without the PVV changing as well. This assumption turns out to be incorrect which somewhat lowers the rationale of having the PVV in the first place.

#### ***Calculating the Program Validation Value***

The algorithm for calculating (or especially *re*-calculating after an alteration) the PVV is kept secret. This violates a fundamental rule of security first made clear by the Dutchman Jean-Guillaume-Hubert-Victor-François-Alexandre-Auguste Kerckhoffs von Nieuwenhof (love that name!) in 1883, that security cannot rely on keeping the method secret. In addition, the algorithm is not really secret as it is installed on every AS/400 in the world; it is only obscure. A second insight by Kerckhoffs was the principle that only cryptographic experts can know the security of a cryptosystem. In today's world it means that only after serious public review of the algorithm can you be reasonably sure that it is secure. The public scrutiny afforded the algorithms competing for the next Federal Encryption Standard (the AES) attests to the acceptance of the importance of Kerckhoffs insight.

Needless to say, no public review of IBM's algorithms has ever taken place. Since the PVV is the last bastion defending the integrity of the system, one might presume some importance be attached to the soundness of the scheme. Back on the CISC-versions of the AS/400, the PVV algorithm was so weak, that



recalculating the PVV after changing a program to system-state was as simple as adding the constant x '7F' twice. Why 7F? You may recall that the internal values of the program state are x'0080' for system state and x'0001' for user state. Their difference is just the x'7F' you had to add. It has proven possible to 'clean-room' re-engineer the algorithm in a couple of hours. As that algorithm has only historic interest today, we'll leave the CISC-system and concentrate on the RISC version.

## The ILE Program Model

The Original Program Model (OPM) was quite simple and straightforward. It has been replaced by the enormously (even hideously) complicated ILE program model (violating yet another Kerckhoffs principle - simplicity). A program can contain one or more modules (each with a different state and/or domain than the program itself), and each module can contain one or more procedures. Anyone of these could have been tampered with, maybe even before they were bound into the final program. So each module and procedure must carry its own checksum and these must all be combined in some fashion into a final PVV for the program as a whole. In the following sections we'll investigate the structure of a program object with the objective of locating the checksums and other information that should go into the PVV calculation.

## Our TEST Program

We'll use SST to disassemble the following (carefully written - albeit very short) program:

```
DCL DD X CHAR(9);
CPYBLA X, "123456789";
```

Following each segment as dumped by SST, we'll try to give an equivalent MI data-declaration of the most important fields as far as we know them and as far as they may be important to the task at hand.

## Object Header

The disassembly starts with the Object Header, actually consisting of two smaller headers, the Segment Header and the pompously named, so-called Encapsulated Program Architecture (EPA) header:

```
MI PROGRAM          SUBTYPE: 01      NAME: TEST                      ADDRESS: 371E7FCE9C 000000
SEGMENT HEADER      (YYSGHDR)
  TYPE 0001 SIZE 0010 NEWFLAGS 00  FLAGS 90  DOMAIN 0001 OBJECT 371E7FCE9C 000000 SPACE 2DC82E457D 001000
371E7FCE9C 000000 0001001000900001 371E7FCE9C000000 7001030000000000 2DC82E457D001000 *.....H.....*
```

```
DCL DD SEGMENT-HEADER CHAR(32);
DCL DD SEGMENT-TYPE                BIN(2) DEF(SEGMENT-HEADER) POS( 1); /* 0001 */
DCL DD SEGMENT-NBR-OF-PAGES       BIN(2) DEF(SEGMENT-HEADER) POS( 3); /* 0010 */
DCL DD SEGMENT-FLAGS              CHAR(2) DEF(SEGMENT-HEADER) POS( 5); /* 0090 */
DCL DD SEGMENT-DOMAIN             BIN(2) DEF(SEGMENT-HEADER) POS( 7); /* 0001 */
DCL DD SEGMENT-OBJECT-ADDR        CHAR(8) DEF(SEGMENT-HEADER) POS( 9); /* 371E...0000 */
DCL DD SEGMENT-MORE-FLAGS         CHAR(8) DEF(SEGMENT-HEADER) POS(17); /* 7001...0000 */
DCL DD SEGMENT-SPACE-ADDR        CHAR(8) DEF(SEGMENT-HEADER) POS(25); /* 2DC8...1000 */
```

We note the object domain value. The address of the associated space at position 25 is actually invalid (V4R4) as it points past the end of the space. This gives rise to an error message after the EPA header, when SST tries to access the space. There is a PTF to fix this bug.

```
EPA HEADER      (YYEPAHDR)
ATT1 80
NAME TEST
SPSZ 00000000
UPSG 0E6792A4C7 000000
RCV2 1000
JPSG 0000000000 000000
OWAU FF1C
GRP 000000000000
COLB 01
DJID 00
AUR 0073
JOPT 00
PBAU FF1C
ACSG 0000000000 000000
ASP 0000
CBSG 0000000000 000000
IPL NUM 00000073
MAXS 0000
LEVL 00000000
DENP 0000000000 000000
JGEN 0000
TYPE 02
SPATT 80
DVER 3600
CTSG 173A0D5C03 000000
PERF 01000000
JID 00000000000000000000
ALIS 0000000000 000000
INFO 0000000000000000
USCNT 0000
POSG 0000000000 000000
TEMP 00000000000000000000
STYP 01
SPIN 00
TIME 10/04/00 14:40:02
OSG 371E7FCE9C 001000
MDTS 10/04/00 14:40:02
GRAU 0000
ATT2 EO
USDAY 0000
DIRP 0000000000 000000
371E7FCE9C 000020 80000201E3C5E2E3 4040404040404040 4040404040404040 *....TEST
371E7FCE9C 000040 404080000000000000 00000030FF1C3600 815D0954F13D8000 0E6792A4C7000000 *.....).1.....G...
371E7FCE9C 000060 000000000000000000 173A0D5C03000000 371E7FCE9C001000 1000000001000000 *.....*
371E7FCE9C 000080 815D09552AFE8000 0000000000000000 0000000000000000 *.....*
371E7FCE9C 0000A0 0000FF1C00000073 0000000000000000 0000000000000000 *.....*
371E7FCE9C 0000C0 0000000000000000 E001000000000000 0000000000000000 *.....*
371E7FCE9C 0000E0 0000000000000000 0000000000000000 0073000000000000 0000000000000000 *.....*
*** ADDRESSING EXCEPTION '81' AT 2DC82E457D 001000
```

```
DCL DD EPA-HEADER CHAR(224);
DCL DD EPA-ATTR1          CHAR(1) DEF(EPA-HEADER) POS( 1); /* 80 */
DCL DD EPA-JOPT           CHAR(1) DEF(EPA-HEADER) POS( 2); /* 00 */
DCL DD EPA-OBJ-TYPE       CHAR(1) DEF(EPA-HEADER) POS( 3); /* 02 *PGM */
DCL DD EPA-OBJ-SUBTYPE    CHAR(1) DEF(EPA-HEADER) POS( 4); /* 01 */
DCL DD EPA-OBJ-NAME       CHAR(30) DEF(EPA-HEADER) POS( 5); /* TEST */
DCL DD EPA-SP-ATTR        CHAR(1) DEF(EPA-HEADER) POS(35); /* 80 */
DCL DD EPA-SP-INITIAL-VALUE CHAR(1) DEF(EPA-HEADER) POS(36); /* 00 */
```

Note the complete object identifier consisting of type/subtype and name. To locate the modules and their procedures we need first the address, **PROGRAM-HDR-ADDR**, of the so-called Program Header.

A very important player is found here, namely the program state. If you change the state you invalidate the PVV. A natural question is: changing what else would invalidate the PVV? Clearly, any of the addresses would not participate in the calculation of the PVV as they can be changed at any time and would be different on a different system, or after a save/restore. But what would? By simple experimentation you can verify that two other fields, the program type and something clumsily called the ‘flags’, actually are important, because changing them also invalidates the PVV. I have outlined and highlighted them below:

## *The AS/400 Security Oracle*

```

Check Object Integrity (CHKOBJITG)

Type choices, press Enter.

User profile . . . . . LSVALGAARD      Name, generic*, *ALL
File to receive output . . . . . xxx      Name
Library . . . . . *LIBL      Name, *LIBL, *CURLIB
Output member options:
  Member to receive output . . . . . *FIRST      Name, *FIRST
  Replace or add records . . . . . *REPLACE      *REPLACE, *ADD
Check domain . . . . . *YES      *YES, *NO
Check program and module . . . . . *YES      *YES, *NO

```

If the message that results from asking the Oracle is:

Command completed successfully.

You know that the PVV is ok. If violations were found, they were written to the file you specified (xxx in the example). You must have the special authority \*AUDIT to run the command. To actually use the Oracle, we must first take care of the little complication we mentioned. But to do this, we need to be a bit further in our analysis. So, let's continue. Here is the MI-structure of the program header. I must remind you that the addresses found in the structures are not MI-pointers. But, using the technique from chapter 7, we can manufacture valid pointers from the addresses:

```
DCL DD PGM-HEADER CHAR(256);
DCL DD PGM-ATTRS CHAR(8) DEF (PGM-HEADER) POS( 1); /* 0000...8004 */
DCL DD PGM-VERSION-TABLE-ADDR CHAR(8) DEF (PGM-HEADER) POS( 9); /* 371E...0280 */
DCL DD PGM-SEGMENT-TABLE-ADDR CHAR(8) DEF (PGM-HEADER) POS(17); /* 371E...1280 */
DCL DD PGM-ACTIVATION-HDR-ADDR CHAR(8) DEF (PGM-HEADER) POS(25); /* 371E...02B0 */
DCL DD PGM-SIGNATURE-TABLE-ADDR CHAR(8) DEF (PGM-HEADER) POS(33); /* 0000...0000 */
DCL DD PGM-STRING-DIRECTORY-ADDR CHAR(8) DEF (PGM-HEADER) POS(41); /* 371E...04F0 */
DCL DD PGM-ACTIVATION-INFO(5) CHAR(8) DEF (PGM-HEADER) POS(49); /* 371E...1200 */
DCL DD PGM-FLAGS CHAR(2) DEF (PGM-HEADER) POS( 89); /* 1000 */
DCL DD PGM-BRING-SIZE CHAR(1) DEF (PGM-HEADER) POS( 91); /* 00 */
DCL DD PGM-HYPERSPACE-ATTRS CHAR(1) DEF (PGM-HEADER) POS( 92); /* 00 */
DCL DD PGM-STATE BIN(2) DEF (PGM-HEADER) POS( 93); /* 0001 */
DCL DD PGM-OBJECT-CHECK-IND CHAR(2) DEF (PGM-HEADER) POS( 95); /* C000 */
DCL DD PGM-SIGNATURE-START CHAR(8) DEF (PGM-HEADER) POS( 97); /* 0000...0000 */
DCL DD PGM-SIGNATURE-END CHAR(8) DEF (PGM-HEADER) POS(105); /* 0000...0000 */
DCL DD PGM-STRING-DIRECTORY-START CHAR(8) DEF (PGM-HEADER) POS(113); /* 371E...04F0 */
DCL DD PGM-STRING-DIRECTORY-END CHAR(8) DEF (PGM-HEADER) POS(121); /* 371E...0523 */
DCL DD PGM-WRI TABLE-HDR-ADDR CHAR(8) DEF (PGM-HEADER) POS(129); /* 371E...0100 */
DCL DD * CHAR(8) DEF (PGM-HEADER) POS(137); /* 0000...0000 */
DCL DD PGM-TYPE CHAR(1) DEF (PGM-HEADER) POS(145); /* 03 */
DCL DD * CHAR(3) DEF (PGM-HEADER) POS(146); /* 0000000 */
DCL DD PGM-ENTRY-PT-MODULE-NBR BIN(4) DEF (PGM-HEADER) POS(149); /* 00000001 */
DCL DD PGM-ENTRY-PT-PROCEDURE-NBR BIN(4) DEF (PGM-HEADER) POS(153); /* 00000001 */
DCL DD PGM-ENTRY-PT-STRING-ID BIN(4) DEF (PGM-HEADER) POS(157); /* 00000010 */
DCL DD PGM-ENTRY-PT-MIN-PARMS BIN(2) DEF (PGM-HEADER) POS(161); /* 0000 */
DCL DD PGM-ENTRY-PT-MAX-PARMS BIN(2) DEF (PGM-HEADER) POS(163); /* 0000 */
DCL DD PGM-PROGRAM-CHECKSUM BIN(4) DEF (PGM-HEADER) POS(165); /* 8727D055 */
DCL DD * CHAR(4) DEF (PGM-HEADER) POS(169); /* 00000000 */
DCL DD PGM-CONVERTED-PGM-INFO CHAR(8) DEF (PGM-HEADER) POS(173); /* 0001...1480 */
DCL DD * CHAR(28) DEF (PGM-HEADER) POS(181); /* 0000...0000 */
DCL DD PGM-HEADER-EXTENSION-ADDR CHAR(8) DEF (PGM-HEADER) POS(209); /* 371E...1100 */
DCL DD PGM-TRACEBACK-ADDR CHAR(8) DEF (PGM-HEADER) POS(217); /* 371E...0440 */
DCL DD PGM-MODULE-TABLE-ADDR CHAR(8) DEF (PGM-HEADER) POS(225); /* 371E...1620 */
DCL DD PGM-OBSERVABILITY-ADDR CHAR(8) DEF (PGM-HEADER) POS(233); /* 371E...1B10 */
DCL DD PGM-MAINTENANCE-HDR-ADDR CHAR(8) DEF (PGM-HEADER) POS(241); /* 371E...1A10 */
DCL DD PGM-EXMAP-ADDR CHAR(8) DEF (PGM-HEADER) POS(249); /* 0000...0000 */
```

I have renamed the second 'FLAG' field to PROGRAM-CHECKSUM because that is what it is. We'll return to this field later. The WRI TABLE-HDR-ADDR advertises that it points to a *writable* portion of the program object. What is going on here? The major reason for the AS/400's stability is not the strength of its design or the quality of its implementation, but very simply that all executable objects are write-protected. Not even a system-state program can alter the code or constants of any other programs directly in memory (setting aside for now the fact that SST and DST can). To understand the issues around this we need to look at how virtual memory (or memory paging) works on the AS/400. The central concept is that of Virtual Effective Address Translation.

## Effective Address Translation

The high-order 12 bits (3 hex digits) determine if the 64-bit effective address is an address to be translated, an E=R (Effective = Real) address, or an E=DS (Effective = Direct-Store) address. If the high-order 3 hex digits of the effective address are 800, the address is a 'real' address. Part of the address space is set aside for these E=R addresses to match the real address range of the machine (which is 52 bits in the current implementation of the PowerPC).

When an E=R address is detected by the hardware, it checks the privilege level bit in the Machine State Register (bit 49) to see whether the process that generated this address can execute privileged instructions



Then the history log:

```

BOUND PROGRAM HISTORY LOG
HI ST LOG SIZE 00000100 ENTRY COUNT 0000000F WRAP COUNT 00 ADDRESS: 371E7FCE9C 000180
NESTING LEVEL 00 EYE CATCHER HI ST CURRENT ENTRY 01
INDEX NUMBER OPERATION CODE NESTING LEVEL TIME STAMP SYSTEM VRM STATUS ASSOCIATED DATA
1 01 00 00 001004144002 0440 00 00 0000000000
2 00 00 00 000000000000 0000 00 00 0000000000
3 00 00 00 000000000000 0000 00 00 0000000000
...
13 00 00 00 000000000000 0000 00 00 0000000000
14 00 00 00 000000000000 0000 00 00 0000000000
15 00 00 00 000000000000 0000 00 00 0000000000
371E7FCE9C 000180 +0000 000001000000000F 000100C8C9E2E300 0101001004144002 0440000000000000 *..... HI ST .....*
371E7FCE9C 0001A0 +0020 0000000000000000 0000000000000000 0000000000000000 0000000000000000 *.....*
6 LINES 371E7FCE9C 0001C0 +0040 TO 371E7FCE9C 000260 +00E0 SAME AS ABOVE

```

Note the ‘eye catcher’ text **HI ST** and the Current Entry count **01**. Let us follow the evolution of the history log as we do things to the program. First the program is created. The log looks like this (operation code for creation is **01**):

```

Address 371E7FCE9C 000180
000180 00000100 0000000F 000100C8 C9E2E300 ..... HI ST.
000190 01010010 04144002 04400000 00000000 ..... 00/10/04 14: 40: 02 0440
0001A0 00000000 00000000 00000000 00000000 .....

```

Now we save the program and restore it some time later. The log within the restored program still carries the history log entry of its creation, but there is now a new entry with operation code **0E** (for restore):

```

000180 00000100 0000000F 000200C8 C9E2E300 ..... HI ST.
000190 01010010 04144002 04400000 00000000 ..... 00/10/04 14: 40: 02 0440
0001A0 0E010010 09142220 04400000 00000000 ..... 00/10/09 14: 22: 20 0440
0001B0 00000000 00000000 00000000 00000000 .....

```

Finally, change the program state at offset x‘00105C’ to system state. The log gets another entry with operation code **03** (altered by SST):

```

000180 00000100 0000000F 000300C8 C9E2E300 ..... HI ST.
000190 01010010 04144002 04400000 00000000 ..... 00/10/04 14: 40: 02 0440
0001A0 0E010010 09142220 04400000 00000000 ..... 00/10/09 14: 22: 20 0440
0001B0 03010010 09143120 04400001 00001050 ..... & 00/10/09 14: 31: 20 0440
0001C0 00000000 00000000 00000000 00000000 .....

```

SST even records the offset (to the nearest 16-byte boundary) of the location of the alteration. Here is the layout of the writable program header:

```

DCL DD WRI TABLE-PGM-HDR CHAR(384);
DCL DD WPH-LOCK-INFO CHAR(128) DEF(WRI TABLE-PGM-HDR) POS( 1);
DCL DD WPH-HISTORY-LOG CHAR(256) DEF(WRI TABLE-PGM-HDR) POS(129);
DCL DD WPH-LOG-SIZE BIN(4) DEF(WPH-HISTORY-LOG) POS( 1); /* 00000100 */
DCL DD WPH-LOG-ENTRIES BIN(4) DEF(WPH-HISTORY-LOG) POS( 5); /* 0000000F */
DCL DD WPH-LOG-WRAP-COUNT CHAR(1) DEF(WPH-HISTORY-LOG) POS( 9); /* 00 */
DCL DD WPH-LOG-CUR-ENTRY CHAR(1) DEF(WPH-HISTORY-LOG) POS(10); /* 03 */
DCL DD * CHAR(1) DEF(WPH-HISTORY-LOG) POS(11); /* 00 */
DCL DD WPH-LOG-EYE-CATCHER CHAR(4) DEF(WPH-HISTORY-LOG) POS(12); /* HI ST */
DCL DD * CHAR(1) DEF(WPH-HISTORY-LOG) POS(16); /* 00 */
DCL DD WPH-LOG-ENTRY(15) CHAR(16) DEF(WPH-HISTORY-LOG) POS(17);

```

And of a single entry:

```

DCL DD HI ST-LOG-ENTRY CHAR(16);
DCL DD HI ST-OPERATION CHAR(1) DEF(HI ST-LOG-ENTRY) POS( 1);
DCL DD HI ST-NESTING-LVL CHAR(1) DEF(HI ST-LOG-ENTRY) POS( 2);
DCL DD HI ST-YYMMDD CHAR(3) DEF(HI ST-LOG-ENTRY) POS( 3);
DCL DD HI ST-HHMMSS CHAR(3) DEF(HI ST-LOG-ENTRY) POS( 6);
DCL DD HI ST-VRM CHAR(2) DEF(HI ST-LOG-ENTRY) POS( 9);
DCL DD * CHAR(2) DEF(HI ST-LOG-ENTRY) POS(11);
DCL DD HI ST-OFFSET BIN(4) DEF(HI ST-LOG-ENTRY) POS(13);

```

## Program Version Table

Not only does SST record in the history log when you altered the program, SST also records in the *Program Version Table*, that the program has been patched. The Program Version Table’s primary purpose is to list the version/release/module information for various components that were used in producing the program. A program is constructed by *binding* together all the modules and procedures it contains. The Binder level field (BN VRM OX LEVEL) is important because it goes into the PVV. The address of Program



Version Table can be found in the Program Header. Before I patched the program, the version table looked like this:

```

PROGRAM VERSION TABLE
TABLE VERSION 00      BN VRM OX LEVEL 02      BN INTERNAL VRM 0360      BN MI VRM 0440
TARGET VRM 0440      CREATE ON VRM 0440      BN EARLIEST VRM 0440      RESERVED 00000000
LANGUAGE VRM 0110      MM MI VRM 0440      OX MI VRM 0360      OX INTERNAL VRM 0370
MM INTERNAL VRM 0360      RESERVED 00000000      CCSID FFFF      LOW OPT 30
HIGH OPT 30      PG MI VRM 0130      PG EARLIEST VRM 0130      PATCH OFFSET 00000000
PATCH SEG INDEX 0000      PATCH ATTRS 00      RESERVED 00
371E7FCE9C 000280 +0000 0002036004400440 0440044000000000 0110044003600370 0360000000000000FF *
371E7FCE9C 0002A0 +0020 001E001E01300130 0000000000000000 0000000000000000

```

I've singled out the *patch information*:

PATCH OFFSET 00000000 PATCH SEG INDEX 0000 PATCH ATTRS 00

After the patch, it looks like this:

```

PROGRAM VERSION TABLE
TABLE VERSION 00      BN VRM OX LEVEL 02      BN INTERNAL VRM 0360      BN MI VRM 0440
TARGET VRM 0440      CREATE ON VRM 0440      BN EARLIEST VRM 0440      RESERVED 00000000
LANGUAGE VRM 0110      MM MI VRM 0440      OX MI VRM 0360      OX INTERNAL VRM 0370
MM INTERNAL VRM 0360      RESERVED 00000000      CCSID FFFF      LOW OPT 30
HIGH OPT 30      PG MI VRM 0130      PG EARLIEST VRM 0130      PATCH OFFSET 00001050
PATCH SEG INDEX 0001      PATCH ATTRS 80      RESERVED 00
371E7FCE9C 000280 +0000 0002036004400440 0440044000000000 0110044003600370 0360000000000000FF *
371E7FCE9C 0002A0 +0020 001E001E01300130 0000105000018000 0000000000000000

```

PATCH OFFSET 00001050 PATCH SEG INDEX 0001 PATCH ATTRS 80

Note the offset of the patch. The actual important information here is the PATCH ATTRS field. If this field has the value x'80' (as set by SST) the program has been patched. The Check Object Integrity program deems a program patched if the patch attribute has the value x'80', even if the PVV is unchanged (say you changed something and changed it back to its original value). Patch the program and patch it back to the original values. Run **CHKOBJI TG** (it will report violations found) and look at the output file:

```

1100900190748AS400  OTEST      LSVALGAARD*PGM      LSVALGAARDPGMMOD

```

If you could reset the patch attributes field to x'00', a patched program would pass **CHKOBJI TG** as long as the PVV is still correct. But the patch information is in the writable first 4K of the program object, so we *can* actually change it. We shall now develop a program to erase the patch information (and the history log with the exception of the first entry, for good measure). First, here is the layout of the Program Version Table:

```

DCL DD PGM-VERSION-TBL CHAR(48);
DCL DD PVT-TABLE-VERSION CHAR(1) DEF(PGM-VERSION-TBL) POS( 1); /* 00 */
DCL DD PVT-BN-VRM-OX-LEVEL CHAR(1) DEF(PGM-VERSION-TBL) POS( 2); /* 02 */
DCL DD PVT-BN-INTERNAL-VRM CHAR(2) DEF(PGM-VERSION-TBL) POS( 3); /* 0360 */
DCL DD PVT-BN-MI-VRM CHAR(2) DEF(PGM-VERSION-TBL) POS( 5); /* 0440 */
DCL DD PVT-TARGET-VRM CHAR(2) DEF(PGM-VERSION-TBL) POS( 7); /* 0440 */
DCL DD PVT-CREATED-ON-VRM CHAR(2) DEF(PGM-VERSION-TBL) POS( 9); /* 0440 */
DCL DD PVT-EARLIEST-VRM CHAR(2) DEF(PGM-VERSION-TBL) POS(11); /* 0440 */
DCL DD * CHAR(4) DEF(PGM-VERSION-TBL) POS(13); /* 00000000 */
DCL DD PVT-LANGUAGE-VRM CHAR(2) DEF(PGM-VERSION-TBL) POS(17); /* 0110 */
DCL DD PVT-MM-VRM CHAR(2) DEF(PGM-VERSION-TBL) POS(19); /* 0440 */
DCL DD PVT-OX-VRM CHAR(2) DEF(PGM-VERSION-TBL) POS(21); /* 0360 */
DCL DD PVT-OX-INTERNAL-VRM CHAR(2) DEF(PGM-VERSION-TBL) POS(23); /* 0370 */
DCL DD PVT-MM-INTERNAL-VRM CHAR(2) DEF(PGM-VERSION-TBL) POS(25); /* 0360 */
DCL DD * CHAR(4) DEF(PGM-VERSION-TBL) POS(27); /* 00000000 */
DCL DD PVT-CCSID CHAR(2) DEF(PGM-VERSION-TBL) POS(29); /* FFFF */
DCL DD PVT-LOW-OPT CHAR(1) DEF(PGM-VERSION-TBL) POS(31); /* 30 */
DCL DD PVT-HIGH-OPT CHAR(1) DEF(PGM-VERSION-TBL) POS(32); /* 30 */
DCL DD PVT-PG-MI-VRM CHAR(2) DEF(PGM-VERSION-TBL) POS(33); /* 0130 */
DCL DD PVT-PG-EARLIEST-VRM CHAR(2) DEF(PGM-VERSION-TBL) POS(35); /* 0130 */
DCL DD PVT-PATCH-OFFSET BIN(4) DEF(PGM-VERSION-TBL) POS(37); /* 00001050 */
DCL DD PVT-PATCH-SEGMENT-INDEX BIN(2) DEF(PGM-VERSION-TBL) POS(39); /* 0001 */
DCL DD PVT-PATCH-ATTRS CHAR(1) DEF(PGM-VERSION-TBL) POS(41); /* 80 */
DCL DD * CHAR(1) DEF(PGM-VERSION-TBL) POS(48); /* 00 */

```

## Program **MI CLNPGM** to Clean Patch Information and History Log

We need access to the encapsulated program object. This is ordinarily forbidden (I can't think of a more appropriate word). But we learned back in chapter 7 how to overcome that problem. The trusty **MI MAKPTR** pointer counterfeiter is just what we need for this. In chapter 7, we developed the **MI TSTPT1** program to access the encapsulated program. We shall build on that foundation.

```

DCL SPCPTR . ARG1 INI T(POINTER);
DCL DD POINTER CHAR(16) BDRY(16);

```

```

DCL PTR .POINTER          DEF(POINTER) POS( 1);

DCL DD PTR-TYPE           CHAR(8) DEF(POINTER) POS( 1);

DCL DD PTR-ADDRESS        CHAR(8) DEF(POINTER) POS( 9);
DCL DD PTR-SEGMENT        CHAR(5) DEF(POINTER) POS( 9); /* REDEF */
DCL DD PTR-OFFSET         CHAR(3) DEF(POINTER) POS(14);

DCL OL      MI MAKPTR (.ARG1) ARG;
DCL SYSPTR .MI MAKPTR;

DCL DD RESOLVE CHAR(34);
DCL DD RESOLVE-TYPE CHAR( 2) DEF(RESOLVE) POS( 1);
DCL DD RESOLVE-NAME CHAR(30) DEF(RESOLVE) POS( 3);
DCL DD RESOLVE-AUTH CHAR( 2) DEF(RESOLVE) POS(33) INIT(X' 0000' );

DCL SPCPTR .PARM1 PARM;
DCL DD PARM-PROGRAM CHAR(10) BAS(.PARM1);

DCL OL PARAMETERS(.PARM1) EXT PARM MIN(1);

DCL SPCPTR .EPA-HEADER;
DCL DD EPA-HEADER CHAR(256) BAS(.EPA-HEADER);
DCL DD EPA-PGM-HEADER-ADDR CHAR(8) DEF(EPA-HEADER) POS(81);

DCL SPCPTR .PROGRAM-HEADER;
DCL DD PROGRAM-HEADER CHAR(256) BAS(.PROGRAM-HEADER);
DCL DD PGM-VERSION-TBL-ADDR CHAR(8) DEF(PROGRAM-HEADER) POS( 9);
DCL DD PGM-WRITABLE-HDR-ADDR CHAR(8) DEF(PROGRAM-HEADER) POS(129);

DCL SPCPTR .PGM-VERSION-TBL;
DCL DD PGM-VERSION-TBL CHAR(48) BAS(.PGM-VERSION-TBL);
DCL DD PVT-PATCH-INFO CHAR(7) DEF(PGM-VERSION-TBL) POS(41);

DCL SPCPTR .HIST-LOG;
DCL DD HIST-LOG CHAR(256) BAS(.HIST-LOG);
DCL DD HIST-SIZE BIN(4) DEF(HIST-LOG) POS( 1);
DCL DD HIST-MAX BIN(4) DEF(HIST-LOG) POS( 5);
DCL DD HIST-COUNTS BIN(2) DEF(HIST-LOG) POS( 9);
DCL DD HIST-ENTRY(15) CHAR(16) DEF(HIST-LOG) POS(17);

DCL DD ENTRY-NBR BIN(4);
DCL DD CUR-ENTRY CHAR(16);
DCL DD CUR-ENTRY-OP-CODE CHAR(1) DEF(CUR-ENTRY) POS( 1);
DCL DD CUR-ENTRY-NESTING CHAR(1) DEF(CUR-ENTRY) POS( 2);
DCL DD CUR-ENTRY-YMDMHS CHAR(6) DEF(CUR-ENTRY) POS( 3);
DCL DD CUR-ENTRY-VRM CHAR(2) DEF(CUR-ENTRY) POS( 9);
DCL DD CUR-ENTRY-STATUS CHAR(1) DEF(CUR-ENTRY) POS(11);
DCL DD CUR-ENTRY-DATA CHAR(5) DEF(CUR-ENTRY) POS(12);

ENTRY * (PARAMETERS) EXT;
CPYBLA RESOLVE-TYPE, X' 0201';
CPYBLAP RESOLVE-NAME, "MI MAKPTR", " ";
RSLVSP .MI MAKPTR, RESOLVE, *, *;

GET-PROGRAM:
CPYBLAP RESOLVE-NAME, PARM-PROGRAM, " ";
RSLVSP .POINTER, RESOLVE, *, *;

```

Now, make a space pointer to the EPA-header starting at offset x'20':

```

GET-EPA-HEADER:
CPYBRAP PTR-OFFSET, X' 20', X' 00';
CALLX .MI MAKPTR, MI MAKPTR, *;
CPYBWP .EPA-HEADER, .POINTER;

```

Use the address of the Program Header to make a pointer to the header. In MI we cannot use addresses, but will have to use pointers:

```

GET-PROGRAM-HEADER:
CPYBLA PTR-ADDRESS, EPA-PGM-HEADER-ADDR;
CALLX .MI MAKPTR, MI MAKPTR, *;
CPYBWP .PROGRAM-HEADER, .POINTER;

```

Similarly, make a space pointer to the Program Version Table, and then to the history log:

```

RESET-PATCH-INFO:
CPYBLA PTR-ADDRESS, PGM-VERSION-TBL-ADDR;
CALLX .MI MAKPTR, MI MAKPTR, *;
CPYBWP .PGM-VERSION-TBL, .POINTER;

```

CPYBREP PVT-PATCH-INFO, X'00';

```

CPYBLA      PTR-ADDRESS,  PGM-WRI TABLE-HDR-ADDR;
ADDLC(S)    PTR-OFFSET,  X' 000080';  /* OFFSET TO HIST LOG */
CALLX       .MI MAKPTR,  MI MAKPTR,  *;
CPYBWP      .HIST-LOG,  .POINTER;

```

```
CPYINV      HI ST-COUNTS, 1;
CPYINV      ENTRY-NBR, 1;
CPYBLA      CUR-ENTRY, HI ST-ENTRY(ENTRY-NBR);
CPYBLA      CUR-ENTRY-OP-CODE, X'01'; /* CREATE */
CPYBLA      CUR-ENTRY-NESTING, X'01'; /* INITIAL */
CPYBLA      CUR-ENTRY-STATUS, X'00'; /* OK */
CPYBREP     CUR-ENTRY-DATA, X'00'; /* NONE */
CPYBLA      HI ST-ENTRY(ENTRY-NBR), CUR-ENTRY;
```

RTX \*.  
/

```
PGM PARM(&PGM) /* CALL ORACLE pgm */
  DCL VAR(&PGM) TYPE(*CHAR) LEN(10)
  DLTF FILE(BADGUYS)
  MONMSG MSGID(CPF2105) /* FILE NOT FOUND */
  STRSST /* ALTER THE PROGRAM */
  CALL MI CLNPGM PARM(&PGM)
  CHKOBJ ITG USRPRF(LSVALGAARD) OUTFILE(BADGUYS)
  DSPPFM FILE(BADGUYS)
  MONMSG MSGID(CPF9812) /* MBR NOT FOUND */
ENDPGM
```

The Program Header points to the Program Maintenance Header. The Maintenance Header has one field of great interest to us. This field is inconspicuously (and rather clumsily) called ‘**FLAGS**’. When I see a 64-bit field called ‘flags’ a **red** flag goes up for me. Something must be going on, and sure enough, what we have here is the *Program Validation Value*! Make a small change to a program and watch the ‘flag’ wave.

```
DCL DD PGM-MAINT-HDR CHAR(256);
DCL DD PMH-PACB-ADDR CHAR(8) DEF(PGM-MAINT-HDR) POS(129); /* 371E...03B0 */
```





```

0017C0 01009000 00000000 371E7FCE 9C001800 .. °....."ôæ..
0017D0 00000000 00000000 00000000 00000000 .....
0017E0 00000000 00000000 00000000 00000000 .....
0017F0 00000000 00000000 00000000 2EAB5B77 ..... ¿$!

```

Calling this a bug or a deliberate obfuscation may depend on one's agenda; in any event, there is significant data here. Change one of the constants and verify for yourself that the data 'in the hole' change accordingly.

Here is the data-declaration for the Module Header:

```

DCL DD MODULE-HEADER CHAR(256);
DCL DD MDH-VERSION CHAR(1) DEF(MODULE-HEADER) POS( 1); /* 00 */
DCL DD MDH-MODULE-TYPE CHAR(1) DEF(MODULE-HEADER) POS( 2); /* 03 */
DCL DD MDH-MODULE-STATE BIN(2) DEF(MODULE-HEADER) POS( 3); /* 0001 */
DCL DD * CHAR(4) DEF(MODULE-HEADER) POS( 5); /* 00000000 */
DCL DD MDH-PROGRAM-HDR-ADDR CHAR(8) DEF(MODULE-HEADER) POS( 9); /* 371E...1000 */
DCL DD MDH-COPYRIGHT-ADDR CHAR(8) DEF(MODULE-HEADER) POS(17); /* 0000...0000 */
DCL DD MDH-SEGMENT-TBL-ADDR CHAR(8) DEF(MODULE-HEADER) POS(25); /* 0000...0000 */
DCL DD MDH-VERSION-TBL-ADDR CHAR(8) DEF(MODULE-HEADER) POS(33); /* 37E1...0470 */
DCL DD * CHAR(24) DEF(MODULE-HEADER) POS(41); /* 00000000 */
DCL DD MDH-OBSERV-INFO-ADDR CHAR(8) DEF(MODULE-HEADER) POS(65); /* 371E...1660 */
DCL DD MDH-STRING-DIR-ADDR CHAR(8) DEF(MODULE-HEADER) POS(73); /* 1EAC...1250 */
DCL DD MDH-ENTRY-POINTS-ADDR CHAR(8) DEF(MODULE-HEADER) POS(81); /* 0000...0000 */
DCL DD * CHAR(16) DEF(MODULE-HEADER) POS(89); /* 00000000 */
DCL DD MDH-PROCEDURE-TBL-ADDR CHAR(8) DEF(MODULE-HEADER) POS(105); /* 37E1...1900 */
DCL DD MDH-CONSTANTS-ADDR CHAR(8) DEF(MODULE-HEADER) POS(113); /* 1EAC...1200 */
DCL DD MDH-EHDT-ADDR CHAR(8) DEF(MODULE-HEADER) POS(121); /* 371E...15A8 */
DCL DD MDH-EHMTLA-ADDR CHAR(8) DEF(MODULE-HEADER) POS(129); /* 371E...1590 */
DCL DD * CHAR(32) DEF(MODULE-HEADER) POS(137); /* 00000000 */
DCL DD MDH-MODULE-ATTRS CHAR(2) DEF(MODULE-HEADER) POS(169); /* 0800 */
DCL DD MDH-OBJECT-ATTRS CHAR(2) DEF(MODULE-HEADER) POS(171); /* 0000 */
DCL DD MDH-ENTRY-PT-DICT-ID BIN(4) DEF(MODULE-HEADER) POS(173); /* 00000007 */
DCL DD MDH-ENTRY-PT-STRING-ID BIN(4) DEF(MODULE-HEADER) POS(177); /* 00000010 */
DCL DD MDH-ENTRY-PT-PROCEDURE BIN(4) DEF(MODULE-HEADER) POS(181); /* 00000001 */
DCL DD MDH-MIN-PARAMETERS BIN(2) DEF(MODULE-HEADER) POS(185); /* 0000 */
DCL DD MDH-MAX-PARAMETERS BIN(2) DEF(MODULE-HEADER) POS(187); /* 0000 */
DCL DD MDH-MAX-MBV-ID-USED BIN(4) DEF(MODULE-HEADER) POS(189); /* 00000003 */
DCL DD MDH-CONSTANTS-BODY? CHAR(1) DEF(MODULE-HEADER) POS(193); /* 01 */
DCL DD MDH-TRANSLATION-REASON CHAR(1) DEF(MODULE-HEADER) POS(194); /* 00 */
DCL DD MDH-PDC-ATTRS CHAR(1) DEF(MODULE-HEADER) POS(195); /* 90 */
DCL DD MDH-HYPERSPACE-ATTRS CHAR(1) DEF(MODULE-HEADER) POS(196); /* 00 */
DCL DD * CHAR(4) DEF(MODULE-HEADER) POS(197); /* 00000000 */
DCL DD MDH-HDR-EXTENSION-ADDR CHAR(8) DEF(MODULE-HEADER) POS(201); /* 371E...1800 */
DCL DD * CHAR(44) DEF(MODULE-HEADER) POS(209); /* 0000...0000 */
DCL DD MDH-MODULE-CHECKSUM CHAR(4) DEF(MODULE-HEADER) POS(253); /* 2EAB5B77 */

```

## Module Version Table

The Module Version Table serves the same purpose as the Program Version Table namely to record the version and release levels of the various components and targets. Both version tables contain a field called the VRM **OX**-level. The significance of the **OX**-levels lies in the fact that if you change the value of either of them from 02 to anything else, you *invalidate* the PVV. Our handy Oracle tells us that. In case you wonder what OX means, here is a clue: Object Translator (X = trans). In fact, the name of the main module that creates the program is "voxlator". Finally, there is also patch information in the Module Version Table:

```

VERSION TABLE          MODULE NBR: 1          ADDRESS: 371E7FCE9C 000470
MODULE: TEST
VERSION                00          MM VRM OX LEVEL 02          LANGUAGE VRM 0110          MM VRM 0440
OX VRM                 0360          MOD INT VRM 0360          INSTRUCTION VRM 0370          TARGET VRM 0440
CREATE ON VRM          0440          OPT LEVEL    001E          CCSID          FFFF
FROM MOD NAME          TEST          FROM MOD QUAL          EARLY COMP VRM 0440          RESERVED 0000
COMPILER NAME          D4E740C3D6D5E5C5D9E3C5C4404040404000          PATCH ATTRS 00
PATCH OFFSET          00000000          PATCH SEG INDEX 0000
RESERVED               00000000000000000000000000000000          0002011004400360 0360037004400440 *
371E7FCE9C 000460 +0000          0002011004400360 0360037004400440 *
371E7FCE9C 000480 +0020          001EFFFFE3C5E2E3 4040404040404040 4040404040404040 *
371E7FCE9C 0004A0 +0040          404000000000000000 000000000000000000 000000000000000000 *
371E7FCE9C 0004C0 +0060          D4E740C3D6D5E5C5 D9E3C5C440404040 4040400004400000 000000000000000000 *
371E7FCE9C 0004E0 +0080          0000000000000000 000000000000000000 000000000000000000 *

```

Here is the data-declaration:

```

DCL DD MOD-VERSION-TBL CHAR(128);
DCL DD MVT-TABLE-VERSION CHAR(1) DEF(MOD-VERSION-TBL) POS( 1); /* 00 */
DCL DD MVT-MM-VRM-OX-LEVEL CHAR(1) DEF(MOD-VERSION-TBL) POS( 2); /* 02 */
DCL DD MVT-LANGUAGE-VRM CHAR(2) DEF(MOD-VERSION-TBL) POS( 3); /* 0110 */
DCL DD MVT-MM-VRM CHAR(2) DEF(MOD-VERSION-TBL) POS( 5); /* 0440 */
DCL DD MVT-OX-VRM CHAR(2) DEF(MOD-VERSION-TBL) POS( 7); /* 0360 */
DCL DD MVT-MM-INTERNAL-VRM CHAR(2) DEF(MOD-VERSION-TBL) POS( 9); /* 0360 */
DCL DD MVT-OX-INTERNAL-VRM CHAR(2) DEF(MOD-VERSION-TBL) POS(11); /* 0370 */

```



```

371E7FCE9C 00149C 00001C F8010008 STD 0, 0X8(1)
371E7FCE9C 0014A0 000020 F82101B8 STD 1, 0X1B8(1)
371E7FCE9C 0014A4 000024 33FF0060 ADDI C 31, 31, 96
371E7FCE9C 0014A8 000028 7C2001C8 TXER 1, 0, 35
371E7FCE9C 0014AC 00002C 419A8053 BCLA 12, 26, 0X8050
371E7FCE9C 0014B0 000030 419D8183 BCLA 12, 29, 0X8180
371E7FCE9C 0014B4 000034 E8820028 LD 4, 0X28(2)
371E7FCE9C 0014B8 000038 38000082 ADDI 0, 0, 130
371E7FCE9C 0014BC 00003C E9820020 LD 12, 0X20(2)
371E7FCE9C 0014C0 000040 980100A8 STB 0, 0XA8(1)
***** EXCEPTION/EXIT HANDLERS ENABLED: 1
371E7FCE9C 0014C4 000044 E8A40010 LD 5, 0X10(4)
371E7FCE9C 0014C8 000048 88040018 LBZ 0, 0X18(4)
371E7FCE9C 0014CC 00004C F8AC0040 STD 5, 0X40(12)
371E7FCE9C 0014D0 000050 980C0048 STB 0, 0X48(12)
371E7FCE9C 0014D4 000054 800100F8 LWZ 0, 0XF8(1)
371E7FCE9C 0014D8 000058 7805FFA3 RLDI CL 5, 0, 63, 62
371E7FCE9C 0014DC 00005C 7C000348 TXER 0, 0, 38
371E7FCE9C 0014E0 000060 419D81C3 BCLA 12, 29, 0X81C0
371E7FCE9C 0014E4 000064 419A8063 BCLA 12, 26, 0X8060
371E7FCE9C 0014E8 000068 38210180 ADDI 1, 1, 384
371E7FCE9C 0014EC 00006C E8010028 LD 0, 0X28(1)
371E7FCE9C 0014F0 000070 7C0803A6 MTSPR 8, 0
371E7FCE9C 0014F4 000074 EBE1FF58 LD 31, 0XFF58(1)
371E7FCE9C 0014F8 000078 4E800021 BCLRL 20, 0

```

## Module Constants

```

MODULE CONSTANTS          MODULE NBR: 1          ADDRESS: 1EAC64A0E7 001200
MODULE: TEST
  SIZE 00000019  RESERVED 00000000000000000000000000000000
1EAC64A0E7 001200 +0000 0000001900000000 000000000000000000 F1F2F3F4F5F6F7F8 F9
*.....123456789*

```

## Primary Associated Space

```

PRIMARY ASSOCIATED SPACE          SPACE SIZE: 4096          ADDRESS: 2DC82E457D 000000
2DC82E457D 000000 +0000 0002000800810001 371E7FCE9C000000 4001000000000000 2DC82E457D001000 *.....H.....*
2DC82E457D 000020 +0020 0000000000000000 0000000000000000 0000000000000000 0000000000000000 *.....*
126 LINES 2DC82E457D 000040 +0040 TO 2DC82E457D 000FE0 +0FE0 SAME AS ABOVE

```

## Which Components Go Into the PVV?

If you are still with me, it is time to reward you for your perseverance. Let's list the values we have identified as having the property that if you change any one of them, the PVV becomes invalid:

Component	Sample value	Size
Procedure Checksum	3C487D28	4
Program Checksum	8727D055	4
Module Checksum	2EAB5B77	4
Program Domain	0001	2
Program State	0001	2
Module State	0001	2
Binder OX Level	02	1
Module OX Level	02	1
(Program Type	03	1)

The last component, the *Program Type*, only invalidates the PVV if you set the value to 05; anything else seems to have no ill effect. As far as I can tell, the program type has this meaning:

- 1 - ILE program
- 2 - ILE service program
- 3 - OPM program
- 4 - Java program

I don't know of any programs of type 05, so we'll ignore the program type for ordinary programs, although it does give me food for thought to contemplate what mysterious secret programs have a type that warrants special treatment when computing the PVV. Ignoring the program type, we are left with eight components for simple programs with only one module and only one procedure. The components have to be combined into a 64-bit (or 8-byte) Validation Value.

## Diffusion and Confusion

When you combine several smaller values into a much larger one, you generally want to "spread" or "fold" the bits over the expanse of the larger receiving value. The process is usually described as a combination of *diffusion*, spreading the bits over the receiver, and *confusion*, disguising the bits such as to make it harder to identify a particular value. Let's look at diffusion first.

Rotating the receiver a number of bits between “adding” in each component will spread or diffuse the incoming bits over the receiver. If we rotate 8 bits, the eight components will just cover the target once; if we rotate 16 bits we get an even better spread because we cover the target twice. The shift amount should not be tied to a byte boundary, *i.e.* be a multiple of eight, because that will concentrate the effect unevenly (often the domain and state values are the same, for instance x ‘0001’) always hitting the same bit position. Better is to add one or more additional shifts, so that the pattern drifts with respect to byte boundaries. The PVV algorithm employs a rotation of 17 bits. The extra 1-bit shift provides confusion.

Additional confusion comes from adding the character values to the target. Addition has the property of losing information; if you add 3 and 5 to get 8, and all you have is the 8, you can’t tell that it was 3 and 5 that were added, it could have been 2 and 6 among others. For additional confusion (with a touch of diffusion), the component bytes are added *backwards*, *i.e.* first byte of component to last byte of target, second byte of component to penultimate byte of target, etc. Addition also helps to diffuse the result as carries can ripple up through the receiver. If we pad each component with binary zeroes at the right to make them all eight bytes long, we can ignore the complication of the addends having different lengths, as adding in zeroes does not change anything. None of this is rocket science and the algorithm produces a rather poor digest in spite of all its antics.

There remains the issue of the starting value of the receiver. Employing a secret starting value provides for further confusion, a good thing. One of four different starting values is used depending on which one of the four combinations of program domain and program state we are dealing with. If this device were not used, the difference between the PVV of the same program in user state and in system state would have been too small (and constant; the old problem with the CISC-version). This valiant attempt to strengthen the algorithm is, unfortunately, marred by the fact that the secret values are poorly chosen with almost constant pair-wise differences, rather than being random.

Assuming then, that we have an algorithm for folding a component onto the receiver, we can still confuse a bit more by varying the *order* in which the components are added. There are  $8! = 40,320$  different orders in which eight components can be folded into a resulting 8-character value. This may look like a lot, but really isn’t. A simple program easily generates the results of all possible permutations. Comparing the result with known values of the PVV reveals in which order the components must be folded onto the receiver. It turns out that there are four different variations or ‘paths’ to follow. Each path has its own order in which to add the components. Which path to use is determined by the data itself; all paths start out the same, but partway through, the partial value of the receiver so far is hashed down to a two-bit selector which selects one of the four paths (“methods”) to follow from now on.

Here is not the place to go further into the painstaking (but fascinating) detailed analysis leading to the algorithm (this chapter is already too long). The following program computes the PVV from given values of the components (those of our **TEST** program).

### **Program *MI PGMVV* Computes the PVV**

To calculate the PVV for a different program, replace the initial values below to suit. Note how each path is given by a sequence of digits, each digit being the ordinal number of a component in the component list held in the COMP array:

```
DCL DD PROCEDURE-CHECKSUM CHAR(4) INIT(X' 3C487D28' );
DCL DD PROGRAM-CHECKSUM   CHAR(4) INIT(X' 8727D055' );
DCL DD MODULE-CHECKSUM    CHAR(4) INIT(X' 2EAB5B77' );
DCL DD PROGRAM-DOMAIN     CHAR(2) INIT(X' 0001' );
DCL DD PROGRAM-STATE      CHAR(2) INIT(X' 0001' );
DCL DD MODULE-STATE       CHAR(2) INIT(X' 0001' );
DCL DD BINDER-LEVEL       CHAR(1) INIT(X' 02' );
DCL DD MODULE-LEVEL       CHAR(1) INIT(X' 02' );

DCL DD COMP(8) CHAR(8);
DCL DD RESULT CHAR(16);
      DCL DD SELECTOR BIN(2) DEF(RESULT) POS(9);

DCL DD METHODS CHAR(8) INIT("1");
      DCL DD METHOD(8) ZND(1,0) DEF(METHODS) POS(1);

DCL DD PATH(0: 3) CHAR(7) INIT
```

```

("2346578", "8367254", "4735826", "3624587");

DCL DD STEP BIN(2);
DCL DD TO BIN(2);
DCL DD FROM BIN(2);
DCL DD THIS BIN(2);

DCL DD SECRET(0:3) CHAR(8) INIT (X' 0E882228A0093A2B' , /* US UD */
                                   X' 1888B20AF012A1EA' , /* SS UD */
                                   X' 17C0597C521E2C8D' , /* US SD */
                                   X' 21C0E95EA227934C' ); /* SS SD */

LOAD-COMPONENTS:
  CPYBLAP COMP(1), PROCEDURE-CHECKSUM, X' 00';
  CPYBLAP COMP(2), PROGRAM-CHECKSUM , X' 00';
  CPYBLAP COMP(3), MODULE-CHECKSUM , X' 00';
  CPYBLAP COMP(4), PROGRAM-DOMAIN , X' 00';
  CPYBLAP COMP(5), PROGRAM-STATE , X' 00';
  CPYBLAP COMP(6), MODULE-STATE , X' 00';
  CPYBLAP COMP(7), BINDER-LEVEL , X' 00';
  CPYBLAP COMP(8), MODULE-LEVEL , X' 00';

SECRET-SEEDING:
  CPYNV SELECTOR, 0; /* US, UD */
  CMPBLA(B) PROGRAM-STATE, X' 0001' /EQ(=+2);
  CPYNV SELECTOR, 1; /* SS, UD */
  CMPBLA(B) PROGRAM-DOMAIN, X' 0001' /EQ(=+2);
  ADDN(S) SELECTOR, 2; /* **, SD */
  CPYBLA RESULT, SECRET(SELECTOR);

  CPYNV STEP, 1;
SELECT:
  CPYBLA RESULT(9:8), RESULT(1:8);
  CPYBTLIS RESULT, RESULT, 17;
  CPYNV THIS, METHOD(STEP);
  CPYBLA RESULT(9:8), COMP(THIS);

  CPYNV TO, 8;
FOLD:
  SUBN FROM, 17, TO;
  ADDLC(S) RESULT(TO:1), RESULT(FROM:1);
  SUBN(SB) TO, 1/POS(FOLD);

  ADDN(S) STEP, 1;
  CMPNV(B) STEP, 8/HI(DONE);
  CMPNV(B) STEP, 2/NEQ(SELECT);

  CPYNV FROM, 8;
  CPYBLA RESULT(9:1), X' 00';
HASH:
  ADDLC(S) RESULT(9:1), RESULT(FROM:1);
  SUBN(SB) FROM, 1/POS(HASH);

  CPYBTRLIS SELECTOR, SELECTOR, 14;
  CPYBLA METHODS(2:7), PATH(SELECTOR);
  B SELECT;

DONE:
  CVTHC MSG-TEXT(1:16), RESULT(1:8);
  CALLI SHOW-MESSAGE, *, .SHOW-MESSAGE;
  RTX *;

```

%INCLUDE SHOWMSG

If you run the program, you get

```

Type reply (if required), press Enter.
From . . . : LSVALLGAARD 10/11/00 22:38:04
E23AB9D56F127989

```

Which is precisely the PVV for **TEST** with the user state attribute. If you change PROGRAM-STATE to system state, x'0080', the result is E2A8D8057879B893. Now change the state and PVV with SST, then run **MI CLNPGM** to remove evidence of the patching.

This program is the result of several people's hard work. It is amazing that this short program of 25 MI-instructions (excluding initialization) duplicates OS/400's intricate calculation. Needless to say, it is not likely that IBM's program for calculating the PVV bears any resemblance to **MI PGMVV**. Calculating the



PVV for more complex programs with several modules will have to iterate over all modules and within each module over all procedures.

## Clearing the History Log on V4R4+

On V4R4 and higher, it turns out that it is not enough to clear the History Log section in the program object. Somehow SST keeps in a secret place (probably the OIR) the information that the program was altered. When you save the object, that secret history information is used. However, it turns out that if you create a duplicate program object (naturally with a *different* name), clean its history log, and then save it, that the clean history log information is used. Then you delete the original and rename the clean copy back to its original name and, *presto*, you've got a clean history log.

If you want to ensure that your program runs under as many releases as possible there are a couple of things to consider. As re-creation of your program from the program template always resets the state attribute to user, you should remove observability. This step prevents re-creation. At the same time, set the *optimize* parameter to \*NO to ensure that the program can run at the earliest release possible:

```

Change Program (CHGPGM)

Type choices, press Enter.

Program . . . . . > TEST          Name, generic*, *ALL
Library . . . . . > *USRLIBL      Name, *USRLIBL
Optimize program . . . . . > *NO   *SAME, *YES, *FULL, *BASIC...
User profile . . . . . *USER      *SAME, *USER, *OWNER
Use adopted authority . . . . . *NO *SAME, *YES, *NO
Remove observable info . . . . . > *ALL *SAME, *ALL, *NONE...
+ for more values
Enable performance collection:
Collection level . . . . . *SAME   *SAME, *NONE, *PEP, *FULL...
Procedures . . . . . *SAME       *ALLPRC, *NONLEAF
Profiling data . . . . . *SAME    *SAME, *NOCOL, *COL, *CLR...
Force program recreation . . . . . *NO *NO, *YES, *NOCRT
Text 'description' . . . . . > 'Test

```

```

Display Program Information

Program . . . . . : TEST          Library . . . . . : LSVALGAARD
Owner . . . . . : LSVALGAARD
Program attribute . . . . . :

Program statistics:
Program size (bytes) . . . . . : 24576
Associated space size (bytes) . . . . . : 4064
Static storage size (bytes) . . . . . : 112
Automatic storage size (bytes) . . . . . : 0
Program state . . . . . : *USER
Program domain . . . . . : *USER
Compiler . . . . . : 5769SS1 V4R4M0
Earliest release that program can run . . . . . : V3R6M0
Conversion required . . . . . : *NO
Sort sequence . . . . . : *HEX
Language identifier . . . . . : *JOBRUN

```

If you do not do this, the earliest release might be the release you compiled under, *e.g.* V4R4M0.

## Digitally Signed Objects

Having recognized the weaknesses in the Program Validation scheme, IBM has recently (V5R1M0) introduced the concept of *signed* objects. A new system value, **QVfy0BJRST**, governs verification of signed objects. This system value specifies the policy to be used for object signature verification during a restore operation. This value applies to objects of types: \*PGM, \*SRVPGM, \*SQLPKG and \*MODULE. It also applies to \*STMF objects which contain Java programs.

If Digital Certificate Manager (OS/400 option 34) is not installed on the system, all objects are treated as unsigned when determining the effects of this system value on those objects during a restore operation. A change to this system value takes effect immediately. The shipped value is 3: Verify object on restore.

These are the possible values:

- 1 Do not verify signatures on restore. Restore all objects regardless of their signature. This value should not be used unless you have signed objects to restore which will fail their signature verification for some acceptable reason.
- 2 Verify signatures on restore. Restore unsigned user-state objects. Restore signed user-state objects, even if the signatures are not valid. This value should be used only if there are specific objects with signatures that are not valid which you want to restore. In general, it is dangerous to restore objects with signatures that are not valid on your system.
- 3 Verify signatures on restore. Restore unsigned user-state objects. Restore signed user-state objects only if the signatures are valid. This value may be used for normal operations, when you expect some of the objects you restore to be unsigned, but you want to ensure that all signed objects have signatures that are valid. This is the default value.
- 4 Verify signatures on restore. Do not restore unsigned user-state objects. Restore signed user-state objects, even if the signatures are not valid. This value should be used only if there are specific objects with signatures that are not valid which you want to restore, but you do not want the possibility of unsigned objects being restored. In general, it is dangerous to restore objects with signatures that are not valid on your system.
- 5 Verify signatures on restore. Do not restore unsigned user-state objects. Restore signed user-state objects only if the signatures are valid. This value is the most restrictive value and should be used when the only objects you want to allow to be restored are those that have been signed by trusted sources. Objects that have the system-state attribute and objects that have the inherit-state attribute are required to have valid signatures. The only value that will allow a system-state or inherit-state object to restore without a valid signature is 1. Allowing such a program represents an integrity risk to your system. If you change the QVFYOBJRST system value to 1 in order to allow such an object to restore onto your system, be sure to change the QVFYOBJRST system value back to its previous value after the object has been restored.

Needless to say, a future chapter on this issue suggests itself. ☺



Blank

## Chapter 16

### User-Defined 5250 Datastream I/O

#### Display Files

A workstation/terminal/display/screen device is treated as a special kind of file, a *device display file*. Like any other file, the file can be either externally described or program described. Display files are ordinarily externally described. You first create a source description (usually in QDDSSRC), then create the display file. A special record format is the *User Defined* format with keyword **USRDFN**:

A	R	<b>USRRCD</b>	<b>USRDFN</b>
---	---	---------------	---------------

This keyword allows (and obligates!) you to build your own datastream to send to the device. When you display the file description (DSPFD *file*), you'll generally see something like this at the bottom of the information:

```
Record Format List
Format      Fields    Record  Format Level  Format
USRRCD      0        0    01A0603D6DA05  USRDFN
Text . . . . .
Total number of formats . . . . . 1
```

Instead of creating your own display file, you can use predefined display files. There is one for 24\*80 displays called **QSN80**, and one for 27\*132 displays called **QSN132**, both in QSYS. Here is part of the device description for QSN80:

```
Display File Description
File Description Header
File . . . . . : FILE      QSN80
Library . . . . . : QSYS
Type of file . . . . . : Device
Device type . . . . . : Display
Auxiliary storage pool ID . . . . . : 01
Device File Attributes
Externally described file . . . . . : Yes
...
Record format level check . . . . . : LVLCHK  *NO
Number of record formats . . . . . : 2
User buffer length . . . . . : 5681
Number of devices . . . . . : 1
Separate indicator area . . . . . : INDARA  NO
Coded character set identifier . . . . . : CCSID  37
Display Attributes Defined in DDS
Keep screen . . . . . : KEEP      Yes
Assume record on display . . . . . : ASSUME  NO
Pass record format . . . . . : PASSRCD NO
Message location specified . . . . . : MSGLOC NO
Print
  Print allowed . . . . . : Yes
  Printer file . . . . . :
  Library . . . . . :
Display size (Lines Positions) . . . . . : DSPSIZ  24 80
Record Format List
(No information available)
```

Somewhat mysteriously, the Record Format list is not available. However, if one dumps the display file object (type/subtype x'1901') and looks at offset x'540', it is evident that two record formats exist in the file and that the list should have been:

```
Record Format List
Format      Fields    Record  Format Level  Format
USRRCD      0        0    0252319D9C405  USRDFN
Text . . . . .
DUMMY       0        0    1FB7F8570FBE7  USRDFN
Text . . . . .
Total number of formats . . . . . 2
```

Here is the dump:

```

Address 199C29B85C 000000
000000 00010008 00808000 199C29B8 5C000000 .....ØØ..æ·½*...
000010 C0010000 00000000 199C29B8 5C000100 {...}.....æ·½*...
000020 80001901 D8E2D5F8 F0404040 40404040 Ø...QSN80
000030 40404040 40404040 40404040 40404040
000040 40408000 00000F00 00000008 38100301 Ø.....
000050 7E12CAA6 B2990000 1CDB5889 53000000 =-w¥r...ûïë...
000060 00000000 00000000 0F7056C0 A9000000 .....Øï{z...

...
Address 199C29B85C 000500
000500 00060001 00000002 00020001 00000000 .....
000510 00010024 000002D0 0000014F 00000107 .....}.....
000520 00000018 00000000 00000000 003E0001 .....
000530 01004400 00010300 01FFFF00 00000000 ..à.....
000540 E4E2D9D9 C3C44040 40400000 00000090 USRRCD .....° ←
000550 C4E4D4D4 E8404040 40400000 000000FC DUMMY .....Û
000560 F0F2F5F2 F3F1F9C4 F9C3F4F0 F5000000 0252319D9C405...
000570 F1C6C2F7 C6F8F5F7 F0C6C2C5 F7000000 1FB7F8570FBE7...
000580 0000006C 0000006C 00000000 80000000 ...%...%...Ø...
000590 00000000 00000000 00000000 001E0000 .....
0005A0 00640000 00000000 00000000 00000000 .Ä.....
0005B0 00C00000 0028FFFF 20000040 00000058 .{.....ï

```

It is convenient to use one of the predefined display files for the purpose of this chapter, namely to revisit the “Hello, World” program and to show how to display the message on the screen using a display file rather than the program message we first used.

## The *MIHWORLD* Program

In chapters 12 and 13 we saw how to build UFCBs and use the SEPT to call the lower level I/O routines. The main purpose of the UFCB was to obtain device independence. This actually works well, as a display is a very different device than a database file, yet we operate on it (almost) the same way:

```

DCL SPCPTR .UFCB INIT(UFCB);
DCL DD UFCB CHAR(210) BDRY(16);
DCL SPCPTR .ODP DEF(UFCB) POS( 1);
DCL SPCPTR .INREC DEF(UFCB) POS( 17); /* input buffer */
DCL SPCPTR .OUTREC DEF(UFCB) POS( 33); /* output buffer */
DCL DD FILE CHAR(10) DEF(UFCB) POS(129) INIT("QSN80");
DCL DD LIB-ID BIN(2) DEF(UFCB) POS(139) INIT(75);
DCL DD LIB CHAR(10) DEF(UFCB) POS(141) INIT("QSYS");
DCL DD MBR-ID BIN(2) DEF(UFCB) POS(151) INIT(-71); /* 1ST */
DCL DD END-LIST BIN(2) DEF(UFCB) POS(209) INIT(32767);

DCL SPCPTR .OPTION INIT(OPTION);
DCL DD OPTION CHAR(4) INIT(X'03000006'); /* PUT, GET, WAIT */

DCL SPCPTR .CONTROL INIT(CONTROL);
DCL DD CONTROL CHAR(14);
DCL DD FORMAT-ITEM CHAR(1) DEF(CONTROL) POS( 1) INIT(X'01');
DCL DD ITEM-LENGTH BIN(2) DEF(CONTROL) POS( 2) INIT(10);
DCL DD RECORD-NAME CHAR(10) DEF(CONTROL) POS( 4) INIT("USRRCD");
DCL DD END-OF-LIST CHAR(1) DEF(CONTROL) POS(14) INIT(X'FF');

DCL DD INREC BAS(.INREC) CHAR(5681);
DCL DD OUTREC BAS(.OUTREC) CHAR(5681);

DCL OL IOLIST (.UFCB, .OPTION, .CONTROL) ARG;
DCL OL OCLIST (.UFCB) ARG;

DCL DD DEVICE-CONTROL-BLOCK BAS(.ODP) BIN(4) POS(17);
DCL DD PUT-GET BAS(.DEVICE-CONTROL) BIN(2) POS(35);
DCL MSPPTR .DEVICE-CONTROL;

DCL SPCPTR @SEPT BASPCO;
DCL SYSPTR .SEPT(6440) BAS(@SEPT);

CALLX .SEPT(12), OCLIST, *; /* open */
CPYBLA OUTREC(1:14), X'0019000373044004110028110C22';
CPYBLA OUTREC(15:12), "Hello, world";
CPYBLA OUTREC(27:4), X'04520000';
ADDSP .DEVICE-CONTROL, .ODP, DEVICE-CONTROL-BLOCK;
CALLX .SEPT(PUT-GET), IOLIST, *;
CALLX .SEPT(11), OCLIST, *; /* close */
RTX *;

```

## Data Management Option List

The *Data Management Option List* specifies the type of I/O request:

```
DCL SPCPTR .OPTION INIT(OPTION);
DCL DD      OPTION CHAR(4) INIT(X'03000006'); /* PUT, GET, WAIT */
```

We may recall from chapter 12, that if the last byte of the fixed, 4-byte structure (the so-called *Device Support* byte) holds an x'06', it means that the operation is a Put-Get request. But in nearly all display file programming, we will need to 'wait' for user input after the 'Put' where we display the screen, but before the 'Get' where we retrieve the user input. That is achieved by setting the value of the first byte, to a x'03', to be 'Put, then Get Next with Wait' operation. Because we use an option to specify the request type, we do not need to set the flags in the UFCB. Actually, I always thought we had to, but experiments indicate that it doesn't matter what the flags are (at least for this case).

## Data Management Control List

This program is the first one where we use a *Data Management Control List* (see chapter 12):

```
DCL SPCPTR .CONTROL INIT(CONTROL);
DCL DD      CONTROL CHAR(14);
DCL DD      FORMAT-ITEM CHAR(1) DEF(CONTROL) POS( 1) INIT(X'01');
DCL DD      ITEM-LENGTH BIN(2) DEF(CONTROL) POS( 2) INIT(10);
DCL DD      RECORD-NAME CHAR(10) DEF(CONTROL) POS( 4) INIT("USRRCD");
DCL DD      END-OF-LIST CHAR(1) DEF(CONTROL) POS(14) INIT(X'FF');
```

The entry ID is x'01', meaning 'Record Format'. Then follows the length (10) of the format name, then the actual format name we are requesting to use (**USRRCD**). The end of the list is signaled by a byte with a value of x'FF'. With this filled in, the Control List is passed to the I/O routine:

```
DCL OL IOLIST (.UFCB, .OPTION, .CONTROL) ARG;
```

## Device Control (or Name) Block

Recall the format of the *Device Control Block* (DCB). The Device Management Entry Point table (the DMEPT) starts 24 bytes into the DCB:

```
DCL SPCPTR .DEV-CONTROL-BLOCK;
DCL SPC      DEV-CONTROL-BLOCK BAS(.DEV-CONTROL-BLOCK);
DCL DD      DCB-MAX-NBR-OF-DEVICES BIN( 2) DIR;
DCL DD      DCB-DEVICES-IN-THE-ODP BIN( 2) DIR;
DCL DD      DCB-DEVICE-NAME CHAR(10) DIR;
DCL DD      DCB-OFFSET-TO-FM-WORK BIN( 4) DIR;
DCL DD      DCB-LENGTH-OF-FM-WORK BIN( 4) DIR;
DCL DD      DCB-INDEX-FOR-LUD-PTR BIN( 2) DIR;
DCL DD      /* DMEPT starts here */
DCL DD      DCB-GET BIN( 2) DIR;
DCL DD      DCB-GET-BY-RRN BIN( 2) DIR;
DCL DD      DCB-GET-BY-KEY BIN( 2) DIR;
DCL DD      * BIN( 2) DIR;
DCL DD      DCB-PUT BIN( 2) DIR;
DCL DD      DCB-PUT-GET BIN( 2) DIR; ← at offset 34
DCL DD      DCB-UPDATE BIN( 2) DIR;
...
```

Just as in chapter 12, we can find the number of the correct SEPT entry (DCB-**PUT-GET**) by using the DEVICE-CONTROL-BLOCK offset in the ODP:

```
DCL DD DEVICE-CONTROL-BLOCK BAS(.ODP) BIN(4) POS(17);
DCL DD PUT-GET BAS(.DEVICE-CONTROL) BIN(2) POS(35);
DCL MSPPTR .DEVICE-CONTROL;

ADDSP      .DEVICE-CONTROL, .ODP, DEVICE-CONTROL-BLOCK;
CALLX      .SEPT(PUT-GET), IOLIST, *;
```

Remember that getting the SEPT entry number this way achieves device independence. Using this method is also required in order for file overrides to work properly.

## Machine Space Pointers

In chapter 12, we used a space pointer to access the DCB, but there are actually two types of pointers that can access data:

- Space Pointers (SPCPTR)
- Machine Space Pointers (**MSPPTR**)

The Machine Space Pointer that we use in this chapter is a more efficient form of pointer. It has no tag bits and is really just an address. The translator can often assign a MSPPTR directly to a hardware register for the life of the entire procedure, thus eliminating repeated register loading and tag bit checking. Because this would negate the usual system protection and checking associated with pointers, there are some restrictions on the use of Machine Space Pointers:

- They cannot be passed as parameters
- They cannot be part of a SPC-structure
- They cannot be based on some other pointer (although you can base something on them)
- Their values do not survive from one invocation (or call) to the next

## Datastream Format

Our program is deceptively simple as it stands because too much is hidden in mysterious hexadecimal constants:

```
CPYBLA      OUTREC(1:14), x'0019000373044004110028110C22';
CPYBLA      OUTREC(15:12), "Hello, world";
CPYBLA      OUTREC(27:4), x'04520000';
```

The above code is placing undifferentiated data directly in the output buffer. But, of course, there is some structure behind it. Let's look at the buffer contents in more detail.

## Buffer Format

The contents of the buffer area sent to the device must have this general format:

```
DCL DD BUFFER CHAR(5681) BAS(*);
  DCL DD BUF-HEADER      CHAR(5) DEF(BUFFER) POS(1);
  DCL DD BUF-SEND-LENGTH BIN(2) DEF(BUF-HEADER) POS(1);
  DCL DD BUF-RECEIVE-LENGTH BIN(2) DEF(BUF-HEADER) POS(3);
  DCL DD BUF-FUNCTION     CHAR(1) DEF(BUF-HEADER) POS(5);
  DCL DD BUF-COMMANDS     CHAR(5676) DEF(BUF-HEADER) POS(6);
```

We see here a *new* construct, BAS(\*), meaning that we are only defining a *structure* that must be based on a real pointer when used. It is not tied to any specific pointer. Here is how we could use the new structure:

```
CPYNV      .OUTREC->BUF-SEND-LENGTH, 25; /* length of command part */
CPYNV      .OUTREC->BUF-RECEIVE-LENGTH, 3;
CPYBLA     .OUTREC->BUF-FUNCTION, x'73'; /* put/get */
CPYBLA     .OUTREC->BUF-COMMANDS, x'044004110028110C22'; /* commands */
```

Note the arrow (->) connecting the basing pointer and the variable of the structure.

The Function code in the buffer header should be x'71' for put only and x'73' for get or put/get requests. It is a bit confusing that the operation code has so many places to go: the flags in the UFCB, the option list, the buffer header, and (yet to come) the buffer commands themselves, but, hey, I didn't design this mess.

## Datastream Commands

Each command starts with an 'escape' character with value x'04'. Calling it an 'escape' character is somewhat confusing because the 'official' EBCDIC escape character is x'27'. Any character can, of course, by convention be used as an escape character with the meaning that the characters which follow have a special meaning; so as long as we have clear in our mind the distinction between 'an' escape character and 'the' Escape character, we're OK. I do remember though, once having to explain the 5250 datastream commands over the telephone and using the term 'escape' character; only hours later, when things still didn't work, did it become clear to the other party that I was not talking about 'the' Escape character.

There can be, and usually are, several commands present in a buffer. Certain commands take parameter bytes as described in the table below, and the 'Write to Display' command can include 'data' to be written to the display. Although this chapter is not meant to be a complete 5250-datastream reference, we'll include sufficient material to enable you to construct useful screens. For a complete reference see SC30-3533-04 or online at: <http://publib.boulder.ibm.com/cgi-bin/bookmgr/bookmgr.cmd/BOOKS/CO2E2001/15.0>

The most common commands are:

Command Description	Value	Parameters	Type
Save Screen	x'02'		
Write to Display	x'11'	C1 C2 Data	Control Characters
Restore Screen	x'12'		
Clear Unit, Alternate	x'20'	U1	Current Position
Write Error Code	x'21'		
Roll	x'23'	R1 R2 R3	Roll Control
Clear Unit	x'40'		
Read Input Fields	x'42'	C1 C2	Control Characters
Clear Format Table	x'50'		
Read MDT Fields	x'52'	C1 C2	Control Characters
Read Whole Screen	x'62'		
Read Immediate	x'72'		

With parameters:

Parameter	Value	Effect
U1	x'00'	Maintain currently defined position
	x'40'	Reset currently defined position
C1	x'00'	No effect
	x'20'	Reset AID and Lock keyboard
	x'40'	Reset AID and Lock keyboard Clear MDT for non-bypass fields
	x'60'	Reset AID and Lock keyboard
		Clear MDT for all fields
	x'80'	Reset AID and Lock keyboard Null non-bypass fields
	x'A0'	Reset AID and Lock keyboard Clear MDT for non-bypass fields Null all fields
	x'C0'	Reset AID and Lock keyboard Clear MDT for non-bypass fields Null non-bypass fields
C2	x'E0'	Reset AID and Lock keyboard Clear MDT for all fields Null all fields
	bit 0    0	Reserved (should be zero)
	bit 1    1	Do not move cursor
	bit 2    1	Cursor blink off
	bit 3    1	Cursor blink on (overrides bit 2)
	bit 4    1	Unlock keyboard, Reset AID byte, and Move cursor to IC address (if any)
	bit 5    1	Sound alarm (beep)
	bit 6    1	Message waiting indicator off
	bit 7    1	Message waiting indicator on (overrides bit 6)

R1	Bit 0	0 1	Roll up Roll down
	Bits 1-2		Reserved (should be zeroes)
	Bits 3-7		Roll the area this number of lines
R2	Bits 0-7		Line number of top line of roll area
R3	Bits 0-7		Line number of bottom line of roll area

The table introduces some new acronyms:

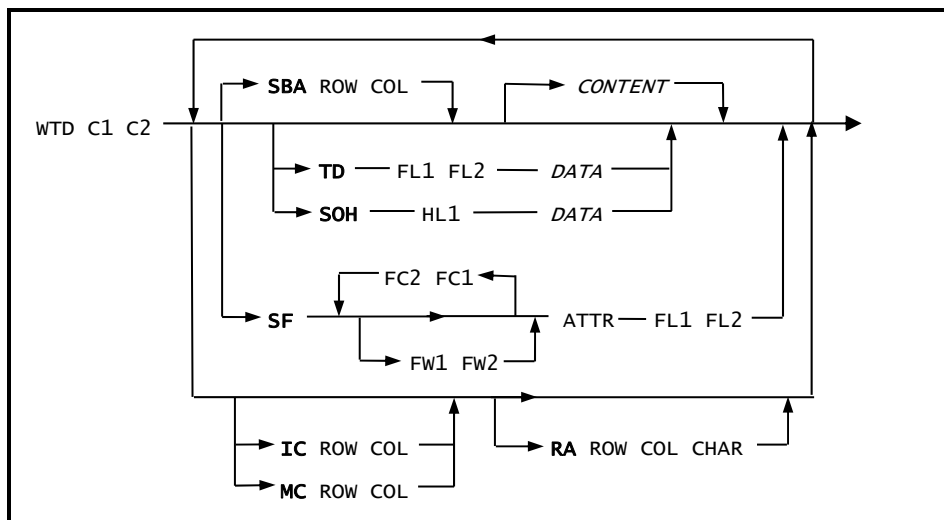
**AID** Attention Identifier: the key that actually returns the screen to the program, *e.g.* the Enter key, a function key, the page down/up keys.

**MDT** Modified Data Tag: is set if anything is typed into a field.

**IC** Insert Cursor order (see below)

## Orders

*Orders* may follow the “Write to Display” command. The data portion of the “Write to Display” command consists of a mixture of orders and ‘user data content’. An order consists of an *Order Introducer* (value less than x'20') followed by various parameters, anything else is considered ‘user data content’, called *CONTENT* in the syntax diagram below:



The following table explains briefly the various orders:

Order Description	Abbr.	Value	Effect
Set Buffer Address	SBA	x'11'	Set current display address to given ROW and COL(umn)
Transparent Data	TD	x'10'	Define arbitrary data at last SBA position. FL1 FL2 two-byte binary field length (not counting length bytes)
Start of Header	SOH	x'01'	Specify format table contents. HL1 one-byte binary header length (not counting length byte)
Start Field	SF	x'1D'	Define input field at last SBA position (points to where the ATTR byte should be written). FC1 FC2 Optional two-byte Field Control word(s) FW1 FW2 Optional two-byte Field Format word

			ATTR Attribute byte FL1 FL2 two-byte binary field length (not counting length bytes)
Insert Cursor	IC	x'13'	Move cursor to ROW, COL at end of Write Command.
Modify cursor	MC	x'14'	Move cursor to ROW, COL for the following data to be written.
Repeat to Address	RA	x'02'	Repeat CHAR(acter) until ROW, COL is reached

### Field Control Word

There can be several optional Field Control Words, controlling more esoteric aspects of the field. They are all characterized by having the high-order bit (bit 0) set in the first character (FC1) of the 2-character control word. That's all we'll have to say about them here.

### Field Format Word

The optional Field Format Word has the two high-order bits set to B'01'. Here is an explanation of the various bits:

Format	Value	Explanation
FW1	Bits 0-1 01	Must be B'01'
	Bit 2 0	Not a Bypass field.
	1	A <i>Bypass</i> field, <i>i.e.</i> no data entry allowed.
	Bit 3 0	DUP key not allowed.
	1	DUP key is allowed.
	Bit 4 0	This field has not been modified (MDT off). This field has been modified (MDT on).
	Bits 5-7	Field edit specifications:
	000	Accept all characters.
	001	Accept only letters , . - and blank.
	010	Accept all characters.
FW2	011	Accept only digits + , . - and blank.
	100	Katakana characters
	101	Reserved (should be zero)
	110	Reject any characters; can be tabbed to.
	111	Accept only digits.
	Bit 0 0	No AUTO Enter.
	1	AUTO Enter when field is filled.
	Bit 1 0	<i>Field Exit Key</i> not required.
	1	<i>Field Exit Key</i> is required.
	Bit 2 0	Accept lower case letters.
	1	Translate operator entered letters to upper case.
	Bit 3 0	Reserved (should be zero).
	Bit 4 0	Not a mandatory entry field.
	1	Is a mandatory entry field.
	Bits 5-7	More field edit specifications:
	000	Don't right adjust the field.
	001	Reserved (should be zero).
	010	Reserved (should be zero).
	011	Reserved (should be zero).
	100	Reserved (should be zero).
	101	Right adjust, fill with zeroes.
	110	Right adjust, fill with blanks.
	111	Mandatory fill by operator.



## Screen Attributes

The screen attribute occupies one position on the screen and governs the visual presentation of the data until the next screen attribute (wrapping around the end of the screen if necessary). The two high-order bits are always B'00'. The attribute position itself has the default background visual presentation with no embellishments. Because of this, there will always be an empty blank space between any two fields. The *CONTENT* can also contain screen attribute characters. These do not define *fields*, but only control the visual presentation of the content to follow. The presentation depends on the capability of the terminal; there are basically three types: monochrome, limited color, and full color. The monochrome terminal can show data in normal and high intensity mapped on the basic color screen to green (or amber) and white, respectively. The full color terminals (and most emulators) support a wider range of colors. This later capability should not be used as a license to make the screen look like a Christmas tree.

Hex Code	No/Limited Color	Full Color
20	Normal	Green
21	Reverse	Green, Reverse
22	High intensity (White)	White
23	High (White), Reverse	White, Reverse
24	Underscore	Green, Underscore
25	Underscore, Reverse	Green, Underscore, Reverse
26	Underscore, High (White)	White, Underscore
27	Non-display	Non-display
28	Blink	Red
29	Blink, Reverse	Red, Reverse
2A	Blink, High (White)	Red, Blink
2B	Blink, High (White), Reverse	Red, Blink, Reverse
2C	Blink, Underscore	Red, Underscore
2D	Blink, Underscore, Reverse	Red, Underscore, Reverse
2E	Blink, Underscore, High (White)	Red, Underscore, Blink
2F	Non-display	Non-display
30	Col Separator	Turquoise, Col Sep.
31	Col Sep., Reverse	Turquoise, Col Sep., Reverse
32	Col Sep., High (White)	Yellow, Col Sep.
33	Col Sep., High (White), Reverse	Yellow, Col Sep., Reverse
34	Col Sep., Underscore	Turquoise, Col Sep., Underscore
35	Col Sep., Underscore, Reverse	Turquoise, Col Sep., Und., Reverse
36	Col Sep., Underscore, High (White)	Yellow, Col Sep., Underscore
37	Col Sep., Non-display	Non-display, Col Sep.
38	Col Sep., Blink	Pink
39	Col Sep., Blink, Reverse	Pink, Reverse
3A	Col Sep., Blink, High (White)	Blue
3B	Col Sep., Blink, High (White), Reverse	Blue, Reverse
3C	Col Sep., Blink, Underscore	Pink, Underscore
3D	Col Sep., Blink, Underscore, Reverse	Pink, Underscore, Reverse
3E	Col Sep., Blink., High (White), Underscore	Blue, Underscore
3F	Col Sep., Non-display	Non-display

The *Column Separators* (little vertical lines between each character) are hardly used anymore and are often suppressed when using an emulator.

## Explanation of the Magic Constants

We can now understand the contents of the buffer sent to the device:

```
CPYBLA      OUTREC(1:14), X'0019000373044004110028110c22';
CPYBLA      OUTREC(15:12), "Hello, world";
CPYBLA      OUTREC(27:4), X'04520000';
```

Send length:   **0019**           25 bytes  
Receive length: **0003**           3 bytes  
Function:       **73**            Put/Get  
Command:        **0440**           Escape, Clear Unit.  
Command:        **04110028**       Escape, Write to Display,  
                                  C1: 00 = nothing,  
                                  C2: 28 = cursor blink off, unlock keyboard, reset AID.  
Order:           **110c22**       Set Buffer Address, ROW: 0C = 12, COL: 22 = 34.  
Content:         "Hello, world"  
Command:        **04520000**       Escape, Read Modified Data Tag fields,  
                                  C1: 00 = nothing,  
                                  C2: 00 = nothing.

Obviously, there is a lot more that can be done with User Defined Data Streams (UDDS) than shown with **MIHWORLD**, but this introduction is really to show you how to begin using UDDS from within MI programs. You may think why would someone go to all the trouble of carefully constructing cryptic hexadecimal streams to define the display, when you can use something much easier, like IBM's Screen Design Aid (SDA) to layout and tweak the display file until you are happy with its layout? Well, for most applications SDA (or directly typing the DDS source) is suitable for the job at hand. However, there are circumstances where SDA and/or hand modified DDS source will not allow you the flexibility to achieve the desired screen layout (or functionality) you are after. In these cases, UDDS is the only option available. But it's not just AS/400 users that hit this situation, in fact the familiar AS/400 tools from IBM such as SDA, SEU, DFU, etc., are all implemented using UDDS.

In the next chapter, I will be presenting a 'wrapper' for screen handling, which will reduce some of the UDDS burden from the programmer using UDDS but still offer the flexibility that UDDS allows.

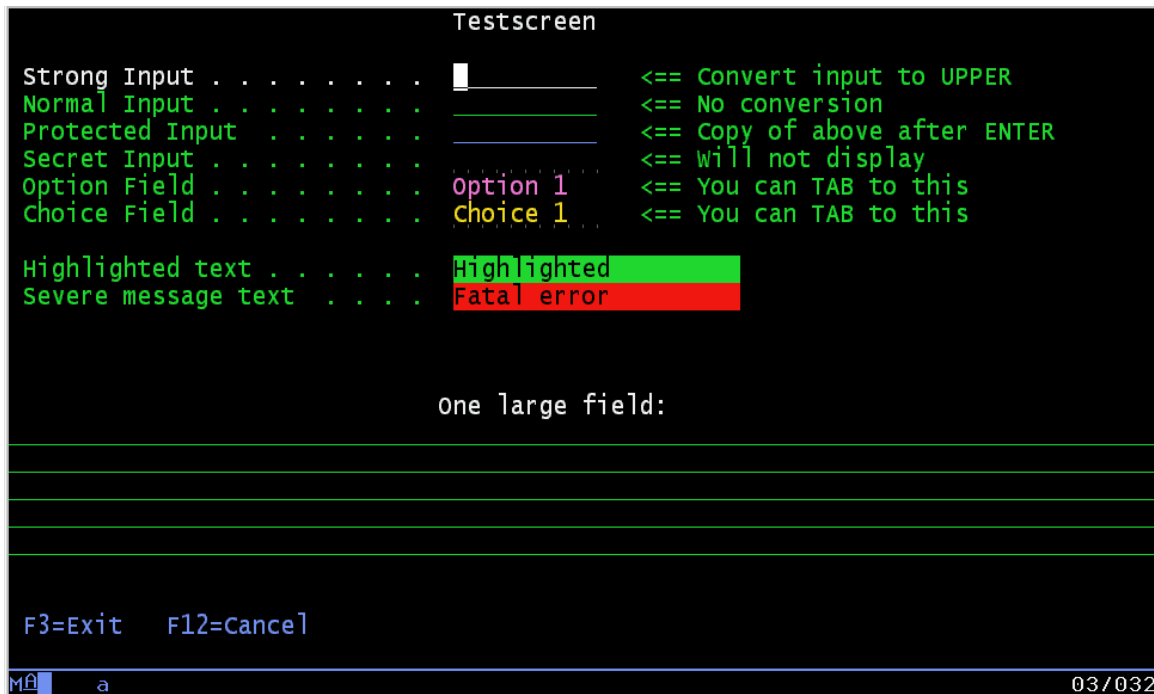
Blank

## Chapter 17

### A Simple Screen I/O Interface

#### Describing Screen Files

A full display file description is extremely complex. Just go and have a look at the **QDFRTVFD** API. Our goal in this chapter is to create a *much* simpler, yet useful, screen interface. The main problem here is to *describe* the screen. Initially we'll do this with an MI data structure. In a later chapter, we'll show how to generate the MI data declaration from an XML (eXtensible Markup Language) description of the screen (the XML description may in turn be generated from an even simpler text-based description). Below is the test screen we'll be working with:



The description of this screen is as shown below. Try to follow along and correlate the visual image with the description. Basically, the description consists of a sequence of *screen items*, where each item starts with a 10-character *introducer* (format described below)

```
DCL DD * CHAR(10) INIT("T010310010")
```

(optionally and usually) followed by a variable length *item value*:

```
DCL DD TITLE CHAR(10) INIT("Testscreen");
```

Introducers and items may be unnamed (using the "\*" convention) or named as needed. If an item is named you can alter its value. After the screen has been received after a read operation, the contents of items that were modified can now be accessed via the named item variables.

Here's the complete screen description:

```
DCL DD * CHAR(10) INIT("T010310010"); DCL DD TITLE CHAR(10) INIT("Testscreen");
DCL DD * CHAR(10) INIT("L010420000");
DCL DD * CHAR(10) INIT("T030010030"); DCL DD * CHAR(30) INIT("Strong Input . . . . .");
DCL DD * CHAR(10) INIT("q030310010"); DCL DD STRONG-INPUT CHAR(10) INIT(" ");
DCL DD * CHAR(10) INIT("L030420030"); DCL DD * CHAR(30) INIT(" <== Convert input to UPPER ");
DCL DD * CHAR(10) INIT("L040010030"); DCL DD * CHAR(30) INIT("Normal Input . . . . .");
DCL DD A-NORMAL-INPUT CHAR(10) INIT("L040310010"); DCL DD NORMAL-INPUT CHAR(10) INIT(" ");
DCL DD * CHAR(10) INIT("L040420030"); DCL DD * CHAR(30) INIT(" <== No conversion ");
DCL DD * CHAR(10) INIT("L050010030"); DCL DD * CHAR(30) INIT("Protected Input ");
DCL DD * CHAR(10) INIT("P050310010"); DCL DD PROTECTED-INPUT CHAR(10) INIT(" ");
```

```

DCL DD * CHAR(10) INIT("L050420030"); DCL DD * CHAR(30) INIT(" <== Copied from above ");
DCL DD * CHAR(10) INIT("L060010030"); DCL DD * CHAR(30) INIT("Secret Input . CHAR(10) INIT(" ");
DCL DD * CHAR(10) INIT("s060310010"); DCL DD SECRET-INPUT CHAR(10) INIT(" ");
DCL DD * CHAR(10) INIT("L060420030"); DCL DD * CHAR(30) INIT(" <== Will not display ");
DCL DD * CHAR(10) INIT("L070010030"); DCL DD * CHAR(30) INIT("Option Field . ");
DCL DD * CHAR(10) INIT("0070310010"); DCL DD OPTION-FIELD CHAR(10) INIT("Option 1");
DCL DD * CHAR(10) INIT("L070420030"); DCL DD * CHAR(30) INIT(" <== You can TAB to this ");
DCL DD * CHAR(10) INIT("L080010030"); DCL DD * CHAR(30) INIT("Choice Field . ");
DCL DD * CHAR(10) INIT("C080310010"); DCL DD CHOICE-FIELD CHAR(10) INIT("Choice 1");
DCL DD * CHAR(10) INIT("L080420030"); DCL DD * CHAR(30) INIT(" <== You can TAB to this ");
DCL DD * CHAR(10) INIT("L100010030"); DCL DD * CHAR(30) INIT("Highlighted text . ");
DCL DD * CHAR(10) INIT("H100310020"); DCL DD HIGHLIGHTED-TEXT CHAR(20) INIT("Highlighted");
DCL DD * CHAR(10) INIT("L100520020"); DCL DD * CHAR(20) INIT(" ");
DCL DD * CHAR(10) INIT("L110010030"); DCL DD * CHAR(30) INIT("Severe message text . ");
DCL DD * CHAR(10) INIT("M110310020"); DCL DD FATAL-ERROR-TEXT CHAR(20) INIT("Fatal error");
DCL DD * CHAR(10) INIT("L110520020"); DCL DD * CHAR(20) INIT(" ");
DCL DD * CHAR(10) INIT("T150300016"); DCL DD * CHAR(16) INIT("One large field: ");
DCL DD * CHAR(10) INIT("L150470000");
DCL DD * CHAR(10) INIT("I150800400"); DCL DD * CHAR(400) INIT(" ");
DCL DD * CHAR(10) INIT("L210010000");
DCL DD * CHAR(10) INIT("U230010079"); DCL DD * CHAR(79) INIT("F3=Exit F12=Cancel ");
DCL DD * CHAR(10) INIT("T240010079"); DCL DD MESSAGE-TEXT CHAR(79) INIT(" ");
DCL DD * CHAR(10) INIT(".000000000");

```

Note the final ‘description terminator’ introducer in the above screen description.

## Item Introducer

The item introducer has following format:

```

DCL MSPPTR .ITEM; /* pointer to the item */
DCL CON ITEM-HEADER-SIZE BIN(2) INIT(10);
DCL DD ITEM CHAR(3574) BAS(.ITEM);
DCL DD ITEM-ATTR CHAR(1) DEF(ITEM) POS( 1);
DCL DD ITEM-ROW ZND(2,0) DEF(ITEM) POS( 2);
DCL DD ITEM-COL ZND(3,0) DEF(ITEM) POS( 4);
DCL DD ITEM-SIZE ZND(4,0) DEF(ITEM) POS( 7);
DCL DD ITEM-DATA CHAR(3564) DEF(ITEM) POS(11); /* 27*132 */

```

Where:

- ITEM-ATTR is a *logical attribute*, a one-character letter-code that determines what kind of item you have, its visual representation, whether it is an input field, and, if so, further characteristics of the field. We take this approach in order to be independent of the actual attribute values that might depend on the precise capabilities of the terminal.
- ITEM-ROW the row position of where the attribute byte should be stored, from 1 and up. The actual field starts at the following character position
- ITEM-COL the column position of where the attribute byte should be stored, from 1 and up. The actual field starts at the following character position Note that the column number is a three-digit number catering for large displays
- ITEM-SIZE the size or length of the item value. This length can vary from zero to a maximum of  $27 \times 132 = 3564$ . Finally, we have the actual field contents to be written (or received on input). If the length was zero, no content is present

## Logical Attributes

There are two kinds of items: output items and input items (strictly speaking, only the latter are *fields* in the narrow sense of a 5250-datastream). We’ll adopt a slighter looser convention and talk about output and input fields on an equal footing. Actually, we’ll introduce yet a third kind of field: the *option* or *choice* field. This is a field you can tab to with the tab-key, but you cannot change the data in the field. (There is a bug in the old 5250 terminal firmware that all emulators faithfully reproduce: even though you cannot enter data into such fields, you can erase their visual contents on the screen with the field-erase key; the field is however not really erased internally.)

Logical attributes define a certain *functional* view of the fields that is only weakly related to their visual appearance. Here is the explanation of the logical attributes:

Attribute	Error	Description
Blank		No attribute; the (output) field starts at the position given
.		End of items. In case you didn't see it, that's a "period"
B		Blank ( <i>i.e.</i> non-display output field)
L		Label; low intensity background output text
T		Title; high intensity output text
H		Highlighted; reverse video output text
U		Informational output text; function keys and options
N		Notification; emphasis (maybe blinking or red)
M		Very strong emphasis; serious error message
P		Protected input; <i>i.e.</i> data that <i>was</i> input, but is now protected
O		Option field that you can tab to and "click on"
C		Option field (or choice field) that you can tab to and "click on"
S		Secure or "secret" input field that will not display data entered
I	E	"Normal" input field; usually underlined
J	F	"Weak" input field; not underlined
Q	G	"Strong" input field; higher intensity, usually underlined
R	K	"Medium" input field; higher intensity, not underlined

Where two attributes are shown (*e.g.* I and E) the second one is the so-called *error* attribute. Using this attribute will highlight the field and "attract" the cursor automatically. Error attributes are transitory in the sense that they are automatically changed back to the non-error attribute for the field after the screen has been shown. Input (and error) fields can be configured to automatically convert data entered from lower case into upper case by using the lower case letter as logical attribute, *e.g.* I instead of I.

## Cursor Position

The screen handler is called with a control block and a data block. The control block specifies a cursor position:

```
DCL SPCPTR .CONTROL INIT(CONTROL);
DCL DD CONTROL CHAR(8);
DCL DD CTRL-OPCODE          CHAR(1) DEF(CONTROL) POS(1);
DCL DD CTRL-CMD-KEY         ZND(2,0) DEF(CONTROL) POS(2);
DCL DD CTRL-CURSOR-POSITION CHAR(5) DEF(CONTROL) POS(4) INIT("00000");
DCL DD CTRL-CURSOR-ROW      ZND(2,0) DEF(CTRL-CURSOR-POSITION) POS(1);
DCL DD CTRL-CURSOR-COL     ZND(3,0) DEF(CTRL-CURSOR-POSITION) POS(3);
```

If one or more error attributes are active the cursor will always go to the first (counted in the order of declaration) of these attributes. If a cursor position (other than 00,000) is given in the control block, the cursor will be positioned and displayed there, otherwise the cursor will default to the position of the first input field (again counted in the order the attributes are declared). When the user presses a key that returns the screen back to our screen handler, the position of where (s)he left the cursor will be returned in the control block. One could further enhance the screen handler by adding to the control block another cursor position field containing the *first* character position within the field.

## The Contiguous Data Block

When declaring data items in MI, these data items are placed in memory in the order they are declared. We can use this to our benefit, as it means that we don't need to explicitly declare all the screen items as part of a structure. As long as we don't 'interrupt' the screen data item declarations with other program variables, we can just lay them out as they come. Initializing a space pointer to the first item declared would suffice to impart addressability to the lot:

```
DCL SPCPTR .SCREEN INIT(SCREEN);
DCL DD SCREEN CHAR(10) INIT("T010310010");
DCL DD TITLE CHAR(10) INIT("Testscreen");
DCL DD * CHAR(10) INIT("L010420000");
...
DCL OL MI SCRNO(.CONTROL, .SCREEN);
```

## The *MI TSTSCR* Test Program

We'll not repeat the data portion; here is the simple code:

```

CPYBLAP      RESOLVE-NAME, "MI SCRNI O", " ";
RSLVSP       . MI SCRNI O, RESOLVE, *, *;

CPYBLA       CTRL-OPCODE, "O"; /* open */
CALLX        . MI SCRNI O, MI SCRNI O, *;

SHOW-SCREEN:
CPYBREP      CTRL-CURSOR-POSITION, "O";
CPYBLA       CTRL-OPCODE, "W"; /* write */
CALLX        . MI SCRNI O, MI SCRNI O, *;
CPYBREP      MESSAGE-TEXT, " ";

CPYBLA       CTRL-OPCODE, "R"; /* read */
CALLX        . MI SCRNI O, MI SCRNI O, *;
CMPNV(B)     CTRL-CMD-KEY, 3/EQ(DONE);
CMPNV(B)     CTRL-CMD-KEY, 12/EQ(DONE);

CPYBLA       PROTECTED-INPUT, NORMAL-INPUT;
CMPBLAP(B)   NORMAL-INPUT, " ", " "/NEQ(=+3);
CPYBLA       A-NORMAL-INPUT, "E"; /* MARK AS ERROR */
CPYBLAP      MESSAGE-TEXT, "Must not be blank", " ";;

B            SHOW-SCREEN;

DONE:
CPYBLA       CTRL-OPCODE, "C"; /* close */
CALLX        . MI SCRNI O, MI SCRNI O, *;
RTX          *;

```

Note the four operation codes: O(pen), W(rite), R(ead), and C(lose). As an example of the use of an error attribute we check if the NORMAL-INPUT field is all blanks. If it is, its attribute is set to the error attribute and a suitable message is placed in MESSAGE-TEXT (which will be displayed in row 24). We keep showing the screen (whether in error or not) until either function key (CTRL-CMD-KEY) 3, or 12 is pressed.

## Command Keys

There are a number of keys that, when pressed, will return the screen contents back to our handler. If the Enter key is pressed it will be returned in the control block as a command key with value zero. Function keys 1 through 24 are returned as key values 1 through 24. Other input terminating keys are returned as follows:

- CMD SHIFTED BACKSP	25
- HELP	26
ROLL DOWN (PGUP)	27
ROLL UP (PGDN)	28
- PRINT	29
- HOME	30
INVALID	31

Keys marked with a hyphen in the above table are rarely used with a PC-based emulator. The only important "special" keys are the roll down/up (page up/down) keys returned as 27/28. Invalid or unknown keys are all mapped to return value 31 (there shouldn't be any, but ...).

## The *MI SCRNI O* Screen Handler

The call-interface is what we just discussed:

```

DCL SPCPTR .P1 PARM;
DCL DD CONTROL CHAR(8) BAS(.P1);
DCL DD OP-CODE CHAR(1) DEF(CONTROL) POS(1);
DCL DD CMD-KEY ZND(2,0) DEF(CONTROL) POS(2);
DCL DD CSR-POSITION CHAR(5) DEF(CONTROL) POS(4);
DCL DD CURSOR-ROW ZND(2,0) DEF(CSR-POSITION) POS(1);
DCL DD CURSOR-COL ZND(3,0) DEF(CSR-POSITION) POS(3);

DCL SPCPTR .P2 PARM;
DCL DD SCREEN CHAR(5000) BAS(.P2);

DCL OL PARMLIST(.P1, .P2) EXT PARM MIN(2);

```

```

/* FORMAT OF SCREEN ITEM: */
DCL CON SCREEN-WIDTH      BIN(2) INIT(80);
DCL CON ITEM-HEADER-SIZE  BIN(2) INIT(10);

DCL MSPPTR . ITEM;
DCL DD ITEM CHAR(3574)     BAS(. ITEM);
DCL DD ITEM-ATTR          CHAR(1) DEF(ITEM) POS( 1);
DCL DD ITEM-ROW           ZND(2,0) DEF(ITEM) POS( 2);
DCL DD ITEM-COL           ZND(3,0) DEF(ITEM) POS( 4);
DCL DD ITEM-SIZE          ZND(4,0) DEF(ITEM) POS( 7);
DCL DD ITEM-DATA          CHAR(3564) DEF(ITEM) POS(11); /* 27*132 */
DCL DD ITEM-POSITION      CHAR(5) DEF(ITEM) POS( 2); /* REDEF */

/* LOGICAL ATTRIBUTES */
/* BLANK NO ATTRIBUTE */
/* . END OF ITEMS (REMAINDER OF STRUCTURE IGNORED) */
/* B BLANK */
/* L LABEL */
/* T TITLE */
/* H HIGHLIGHTED */
/* U INFORMATIONAL */
/* M MESSAGE (STRONG) */
/* P PROTECTED (INPUT) */
/* O OPTION - CAN TAB TO */
/* C CHOICE - CAN TAB TO */
/* S INPUT (SECURE) - NON-DISPLAY s ANY CASE */
/* I E INPUT (NORMAL) - GREEN, UNDERLINED i e ANY CASE */
/* J F INPUT (WEAK ) - GREEN, NO UNDERLINING j f ANY CASE */
/* Q G INPUT (STRONG) - WHITE, UNDERLINED q g ANY CASE */
/* R K INPUT (MEDIUM) - WHITE, NO UNDERLINING r k ANY CASE */

```

## File control Block and Data Management Entries

I/O will be done using SEPT-calls as discussed in chapters 12 and 16:

```

DCL SPCPTR @SEPT BASPCO;
DCL SPCPTR . SEPT(6440) BAS(@SEPT);

DCL SPCPTR . UFCB INIT(UFCB);
DCL DD UFCB CHAR(210) BDRY(16);
DCL SPCPTR . ODP DEF(UFCB) POS( 1);
DCL SPCPTR . INBUF DEF(UFCB) POS( 17);
DCL SPCPTR . OUTBUF DEF(UFCB) POS( 33);
DCL SPCPTR . IO-FEEDBACK DEF(UFCB) POS( 65);

DCL DD FILE-NAME CHAR(10) DEF(UFCB) POS(129) INIT("QSN80");
DCL DD LIB-ID BIN ( 2) DEF(UFCB) POS(139) INIT( 72);
DCL DD LIBRARY CHAR(10) DEF(UFCB) POS(141) INIT("QSYS");
DCL DD MBR-ID BIN ( 2) DEF(UFCB) POS(151) INIT(-73);
DCL DD MEMBER CHAR(10) DEF(UFCB) POS(153) INIT("FIRST");

DCL DD FLAGS-1 CHAR( 1) DEF(UFCB) POS(175) INIT(X' 80');
DCL DD FLAGS-2 CHAR( 1) DEF(UFCB) POS(176) INIT(X' 00');

DCL DD NO-MORE-PARAMS BIN ( 2) DEF(UFCB) POS(209) INIT(32767);

```

After the first I/O-operation, the IO-FEEDBACK contains the following information:

```

DCL SPC IO-FEEDBACK BAS(. IO-FEEDBACK);
DCL DD OFFSET-TO-DEV BIN ( 2) DIR;
DCL DD NBR-OF-PUTS BIN ( 4) DIR;
DCL DD NBR-OF-GETS BIN ( 4) DIR;
DCL DD NBR-OF-PUTGETS BIN ( 4) DIR;
DCL DD NBR-OF-OTHER-OP BIN ( 4) DIR;
DCL DD CUR-OPERATION CHAR( 1) DIR;
DCL DD PREV-OPERATION CHAR( 1) DIR;
DCL DD REC-FORMAT-NAME CHAR(10) DIR;
DCL DD ACTUAL-DEV-TYPE CHAR( 1) DIR;
DCL DD ACTUAL-DEV-CLAS CHAR( 1) DIR;
DCL DD ACTUAL-DEV-NAME CHAR(10) DIR;
DCL DD ACTUAL-REC-SIZE BIN ( 4) DIR;
DCL DD REQUEST-ID CHAR(80) DIR;
DCL DD NBR-OF-BLOCKS BIN ( 2) DIR;
DCL DD * BIN ( 4) DIR;
DCL DD CUR-BLOCK-COUNT BIN ( 4) DIR;

```

Finding the SEPT-entries to use is done via the Data Management Device list:



```

DCL SPC ODP BAS(. ODP);
DCL DD * CHAR(16) DIR;
DCL DD DEV-OFFSET BIN ( 4) DIR;

DCL SPCPTR . DMDEV;
DCL SPC FUNCTION-LIST BAS(. DMDEV);
DCL DD MAX-DEVICE BIN ( 2) DIR;
DCL DD NBR-DEVICES BIN ( 2) DIR;
DCL DD DEVICE-NAME CHAR(10) DIR;
DCL DD WORKAREA-OFFSET BIN ( 4) DIR;
DCL DD WORKAREA-LENGTH BIN ( 4) DIR;
DCL DD LUD-PTR-INDEX BIN ( 2) DIR;

DCL DD GET-FCT BIN ( 2) DIR;
DCL DD GET-DIR BIN ( 2) DIR;
DCL DD GET-KEY BIN ( 2) DIR;
DCL DD * BIN ( 2) DIR;
DCL DD PUT-FCT BIN ( 2) DIR;
DCL DD PUT-GET BIN ( 2) DIR;

```

Here are the Data Management Option and Control lists. Note that we use the standard User-Defined-Datastream (record format **USRRCD**) for the **QSN80** display file:

```

DCL SPCPTR . PUTOPT INIT(PUTOPT);
DCL DD PUTOPT CHAR(4);
DCL DD PUT-OPTION-BYTE CHAR(1) DEF(PUTOPT) POS(1) INIT(X' 00' );
DCL DD PUT-SHARE-BYTE CHAR(1) DEF(PUTOPT) POS(2) INIT(X' 00' );
DCL DD PUT-DATA-BYTE CHAR(1) DEF(PUTOPT) POS(3) INIT(X' 00' );
DCL DD PUT-DEVICE-BYTE CHAR(1) DEF(PUTOPT) POS(4) INIT(X' 05' );

DCL SPCPTR . GETOPT INIT(GETOPT);
DCL DD GETOPT CHAR(4);
DCL DD GET-OPTION-BYTE CHAR(1) DEF(GETOPT) POS(1) INIT(X' 03' );
DCL DD GET-SHARE-BYTE CHAR(1) DEF(GETOPT) POS(2) INIT(X' 00' );
DCL DD GET-DATA-BYTE CHAR(1) DEF(GETOPT) POS(3) INIT(X' 00' );
DCL DD GET-DEVICE-BYTE CHAR(1) DEF(GETOPT) POS(4) INIT(X' 06' );

DCL SPCPTR . FORMAT INIT(FORMAT);
DCL DD FORMAT CHAR(14);
DCL DD FORMAT-ID CHAR( 1) DEF(FORMAT) POS( 1) INIT(X' 01' );
DCL DD FORMAT-SIZE BIN ( 2) DEF(FORMAT) POS( 2) INIT( 10 );
DCL DD FORMAT-NAME CHAR(10) DEF(FORMAT) POS( 4) INIT("USRRCD");
DCL DD FORMAT-END CHAR( 1) DEF(FORMAT) POS(14) INIT(X' FF' );

DCL OL OPEN (. UFCB);
DCL OL PUT (. UFCB, . PUTOPT, . FORMAT);
DCL OL GET (. UFCB, . GETOPT, . FORMAT);
DCL OL CLOSE(. UFCB);

```

## Buffer Formats

We use three different formats, one to clear the display unit, one to put a datastream, and one to get all modified input fields:

```

DCL DD BUFFER CHAR(5014) BAS(*);
DCL DD IO-COMMAND CHAR(5014) DEF(BUFFER) POS( 1);
DCL DD P-HEADER CHAR( 14) DEF(BUFFER) POS( 1); /*REDEF*/
DCL DD DATA-STREAM CHAR(5000) DEF(BUFFER) POS(15);

DCL DD INIT-COMMAND CHAR(7);
DCL DD INIT-SEND-L BIN( 2) DEF(INIT-COMMAND) POS(1) INIT(0002);
DCL DD INIT-RECV-L BIN( 2) DEF(INIT-COMMAND) POS(3) INIT(0000);
DCL DD INIT-FUNCT CHAR(1) DEF(INIT-COMMAND) POS(5) INIT(X' 71' );

DCL DD INIT-ESC1 CHAR(1) DEF(INIT-COMMAND) POS(6) INIT(X' 04' );
DCL DD INIT-CLR-UNIT CHAR(1) DEF(INIT-COMMAND) POS(7) INIT(X' 40' );

DCL DD PUT-HEADER CHAR(14);
DCL DD PUT-SEND-L BIN( 2) DEF(PUT-HEADER) POS( 1);
DCL DD PUT-RECV-L BIN( 2) DEF(PUT-HEADER) POS( 3) INIT(0000);
DCL DD PUT-FUNCT CHAR(1) DEF(PUT-HEADER) POS( 5) INIT(X' 71' );

DCL DD PUT-ESC1 CHAR(1) DEF(PUT-HEADER) POS( 6) INIT(X' 04' );
DCL DD PUT-CLR-FMT CHAR(1) DEF(PUT-HEADER) POS( 7) INIT(X' 50' );

DCL DD PUT-ESC2 CHAR(1) DEF(PUT-HEADER) POS( 8) INIT(X' 04' );
DCL DD PUT-WRITE CHAR(1) DEF(PUT-HEADER) POS( 9) INIT(X' 11' );

DCL DD PUT-CC CHAR(2) DEF(PUT-HEADER) POS(10);

```

```

DCL DD KB-RESET CHAR(1) DEF(PUT-CC) POS( 1) INIT(X' 60' );
DCL DD BLNK-OFF1 CHAR(1) DEF(PUT-CC) POS( 2) INIT(X' 20' );

DCL DD PUT-IC CHAR(1) DEF(PUT-HEADER) POS(12) INIT(X' 13' );
DCL DD PUT-ROW CHAR(1) DEF(PUT-HEADER) POS(13);
DCL DD PUT-COL CHAR(1) DEF(PUT-HEADER) POS(14);

DCL DD GET-MDT CHAR(13);
DCL DD MDT-SEND-L BIN( 2) DEF(GET-MDT) POS( 1) INIT(0008);
DCL DD MDT-RECV-L BIN( 2) DEF(GET-MDT) POS( 3) INIT(5000);
DCL DD MDT-FUNCT CHAR(1) DEF(GET-MDT) POS( 5) INIT(X' 73' );

DCL DD MDT-ESC1 CHAR(1) DEF(GET-MDT) POS( 6) INIT(X' 04' );
DCL DD MDT-WRITE CHAR(1) DEF(GET-MDT) POS( 7) INIT(X' 11' );

DCL DD MDT-WRT-CC CHAR(2) DEF(GET-MDT) POS( 8);
DCL DD * CHAR(1) DEF(MDT-WRT-CC) POS( 1) INIT(X' 00' );
DCL DD BLNK-ON CHAR(1) DEF(MDT-WRT-CC) POS( 2) INIT(X' 18' );

DCL DD MDT-ESC2 CHAR(1) DEF(GET-MDT) POS(10) INIT(X' 04' );
DCL DD MDT-READ CHAR(1) DEF(GET-MDT) POS(11) INIT(X' 52' );
DCL DD MDT-RD-CC CHAR(2) DEF(GET-MDT) POS(12);
DCL DD KB-LOCK CHAR(1) DEF(MDT-RD-CC) POS( 1) INIT(X' 20' );
DCL DD BLNK-OFF2 CHAR(1) DEF(MDT-RD-CC) POS( 2) INIT(X' 20' );

```

## Orders

Every screen item is translated into an SBA (Set Buffer Address) order to the position given in the item:

```

DCL DD SBA-ORDER CHAR(3);
DCL DD SBA CHAR(1) DEF(SBA-ORDER) POS(1) INIT(X' 11' );
DCL DD SBA-ROW CHAR(1) DEF(SBA-ORDER) POS(2);
DCL DD SBA-COL CHAR(1) DEF(SBA-ORDER) POS(3);

```

If the item is an input or option field, an SF-order is placed in the buffer:

```

DCL DD SF-ORDER CHAR(6);
DCL DD SF CHAR(1) DEF(SF-ORDER) POS(1) INIT(X' 1D' );
DCL DD SF-FW CHAR(2) DEF(SF-ORDER) POS(2);
DCL DD SF-ATTR CHAR(1) DEF(SF-ORDER) POS(4);
DCL DD SF-SIZE BIN(2) DEF(SF-ORDER) POS(5);

```

## Command Key Translation

We have chosen to translate the *raw* AID keys into a more sensible range of values spanning the interval [0:31]. The following table defines the translated values and also assigns the value 31 (i.e. x 1F') to all unknown key values:

```

DCL DD CMD-KEY-TABLE CHAR(256);
DCL DD CMD-KEY-MAP(16) CHAR(16) DEF(CMD-KEY-TABLE) POS(1) INIT
(X' 1F1F1F1F1F1F1F1F1F1F1F1F1F1F1F1F' , /* 00: 0F */
X' 1F1F1F1F1F1F1F1F1F1F1F1F1F1F1F1F' , /* 10: 1F */
X' 1F1F1F1F1F1F1F1F1F1F1F1F1F1F1F1F' , /* 20: 2F */
X' 1F0102030405060708090A0B0C1F1F1F' , /* 30: 3F * CMD 01-12 */
X' 1F1F1F1F1F1F1F1F1F1F1F1F1F1F1F1F' , /* 40: 4F */
X' 1F1F1F1F1F1F1F1F1F1F1F1F1F1F1F1F' , /* 50: 5F */
X' 1F1F1F1F1F1F1F1F1F1F1F1F1F1F1F1F' , /* 60: 6F */
X' 1F1F1F1F1F1F1F1F1F1F1F1F1F1F1F1F' , /* 70: 7F */
X' 1F1F1F1F1F1F1F1F1F1F1F1F1F1F1F1F' , /* 80: 8F */
X' 1F1F1F1F1F1F1F1F1F1F1F1F1F1F1F1F' , /* 90: 9F */
X' 1F1F1F1F1F1F1F1F1F1F1F1F1F1F1F1F' , /* A0: AF */
X' 1F0D0E0F101112131415161718191F1F' , /* B0: BF * CMD 13-24 */
X' 1F1F1F1F1F1F1F1F1F1F1F1F1F1F1F1F' , /* C0: CF */
X' 1F1F1F1F1F1F1F1F1F1F1F1F1F1F1F1F' , /* D0: DF */
X' 1F1F1F1F1F1F1F1F1F1F1F1F1F1F1F1F' , /* E0: EF */
X' 1F001F1A1B1C1D1F1E1F1F1F1F1F1F1F' ); /* F0: FF * SPECIAL */

/* F1 ENTER PF KEY 00 */
/* 31: 3C FUNCTION KEY 01-12 PF KEY 01-12 */
/* B1: BC FUNCTION KEY 13-24 PF KEY 13-24 */
/* BD - CMD SHIFTED BACKSP PF KEY 25 */
/* F3 - HELP PF KEY 26 */
/* F4 ROLL DOWN (PGUP) PF KEY 27 */
/* F5 ROLL UP (PGDN) PF KEY 28 */
/* F6 - PRINT PF KEY 29 */
/* F8 - HOME PF KEY 30 */
/* OTHERS INVALID PF KEY 31

```

## Data Content Translation

Non-printable characters are translated into a question mark (x'6F'). Of more interest is that blanks (x'40') are translated into null characters (x'00'). The rationale for this is that insertion of data in the middle of a field means that characters at the end of the field are dropped off. Only null characters can be dropped without a 'No Room To Insert Data' field error condition arising:

[illegible]

## Attribute Translation

The attribute letter at the start of every item introducer is translated into a real attribute using this table:

DCL	DD ATTR=TABLE CHAR(256);
DCL	DD *(16) CHAR(16) DEF(ATTR=TABLE)
(X'	20202020202020202020202020202020' POS(1) INI T
X'	20202020202020202020202020202020' /* 00-0F */
X'	20202020202020202020202020202020' /* 10-1F */
X'	20202020202020202020202020202020' /* 20-2F */
X'	20202020202020202020202020202020' /* 30-3F */
X'	00202020202020202020202020202020' /* 40-4F SPACE => NULL */
X'	20202020202020202020202020202020' /* 50-5F */
X'	20202020202020202020202020202020' /* 60-6F */
X'	20202020202020202020202020202020' /* 70-7F */
X'	2020E7F220E5E1E361E4202020202020' /* 80-8F ..bc.efghi */
X'	20E0EF3606968F874E6E2202020202020' /* 90-9F .jklmnopqr */
X'	2020F7627A202020202020202020202020' /* A0-AF ..stu.... */
X'	20202020202020202020202020202020' /* B0-BF */
X'	202027B220A5A1A321A420202020202020' /* C0-CF ..BC.EFGHI */
X'	20A0A3202928B83EA6A220202020202020' /* D0-DF ..JKLMNOPQR */
X'	2020BF223A202020202020202020202020' /* E0-EF ..STU.... */
X'	20202020202020202020202020202020' /* F0-FF */

Actual attribute characters are less than x'40', i.e. are described using the low-order six bits only. The remaining two bits of the translated value are used as follows:

Bit 0:	0	an output field
	1	an input or option field
Bit 1:	0	do <i>not</i> translate input to upper case
	1	do translate input to upper case

These two bits are, of course, masked off before the attribute character is stored in the buffer.

### Current Cursor Position

We start a Write operation by zeroing the following two positioning variables:

```

DCL DD CURSOR-POSITIONS CHAR(10);
      DCL DD THE-POSITION CHAR(5) DEF(CURSOR-POSITIONS) POS(1);
          DCL DD THE-ROW ZND(2,0) DEF(THE-POSITION) POS(1);
              DCL DD THE-COL ZND(3,0) DEF(THE-POSITION) POS(3);

      DCL DD ERR-POSITION CHAR(5) DEF(CURSOR-POSITIONS) POS(6);
          DCL DD ERR-ROW ZND(2,0) DEF(ERR-POSITION) POS(1);
              DCL DD ERR-COL ZND(3,0) DEF(ERR-POSITION) POS(3);

```

When the first (if any) input field is encountered while scanning the screen items from beginning to end, its position is stored in THE-POSITION. When the first (if any) error field is encountered, *its* position is stored in ERR-POSITION. This allows us to determine where the cursor should go after the screen has been written. When the user presses an AID key, the cursor position and the AID key are returned at the beginning of the datastream sent from the device:

```
DCL DD DATA-RETURNED CHAR(5000) BAS(*);
DCL DD ROW-POS CHAR(1) DEF(DATA-RETURNED) POS(1);
DCL DD COL-POS CHAR(1) DEF(DATA-RETURNED) POS(2);
DCL DD AID-KEY CHAR(1) DEF(DATA-RETURNED) POS(3);
```

Finally, a few more working variables:

```
DCL DD ATTR CHAR(1);
DCL DD FIELD-SIZE BIN(2);
DCL DD LENGTH BIN(2);
DCL DD GOT BIN(2);
DCL DD P BIN(2); /* subscript to where to place data in output stream */
```

```
DCL MSPPTR .INPUT;
DCL DD .INPUT CHAR(5000) BAS(.INPUT);
```

The following declaration provides a convenient vehicle for converting a character to a two-byte binary number (and vice versa):

```
DCL DD BINARY-VALUE BIN(2) INIT(0);
DCL DD BIN-CHAR CHAR(1) DEF(BINARY-VALUE) POS(2);
```

## The Screen Handler Code

The complete code can be found under the name **MI SCRNI O** in the “downloadable programs section”. First we branch out based on the operation code (invalid codes have no effect):

```
ENTRY * (PARMLIST) EXT;
  CMPBLA(B) OP-CODE, "W"/EQ(WRITE-TERMINAL);
  CMPBLA(B) OP-CODE, "R"/EQ(READ-TERMINAL);
  CMPBLA(B) OP-CODE, "C"/EQ(CLOSE-TERMINAL);
  CMPBLA(B) OP-CODE, "O"/EQ(OPEN-TERMINAL);
  B DONE;
```

## Open Terminal

As in chapter 12, we find the Data Management Entry Point Table via the Open Data Path (the ODP):

```
OPEN-TERMINAL:
  CALLX .SEPT(12), OPEN, *;
  ADDSPP .DMDEV, .ODP, DEV-OFFSET;
```

Then we clear the display unit and its format table (removing all input fields):

```
CLEAR-DISPLAY-UNIT:
  CPYBLA .OUTBUF->IO-COMMAND, INIT-COMMAND;
  CALLX .SEPT(PUT-FCT), PUT, *;
  B DONE;
```

## Close Terminal

```
CLOSE-TERMINAL:
  CALLX .SEPT(11), CLOSE, *;
  B DONE;
```

## Write to Terminal

After having cleared the cursor position tracking variables, we start with the first screen item:

```
WRITE-TERMINAL:
  CPYBREP CURSOR-POSITIONS, "O";
  SETSPP .ITEM, SCREEN;
  CPYNV P, 1;
```

If the item attribute is a point (“.”), we are done, otherwise translate the attribute and make a binary copy of the (zoned) item size variable (because we cannot use a zoned value as a subscript):

```

WRI TE-NEXT-I TEM:
  CMPBLA(B)    I TEM-ATTR, " " /EQ(SET-CURSOR);
  XLATEWT      ATTR, I TEM-ATTR, ATTR-TABLE;
  CPYNV        F IELD-SI ZE, I TEM-SI ZE;

```

Convert the (zoned) buffer address to binary single-byte values, place them in an SBA order and write the order to the output buffer:

```

WRI TE-SBA-ORDER:
  CPYNV        B I NARY-VALUE, I TEM-ROW;
  CPYBLA       SBA-ROW, B I N-CHAR;
  CPYNV        B I NARY-VALUE, I TEM-COL;
  CPYBLA       SBA-COL, B I N-CHAR;
  CPYBLA       . OUTBUF->DATA-STREAM(P: 3), SBA-ORDER;
  ADDN(S)      P, 3; /* bump output address past the SBA */

```

If the high-order bit is set, the translated attribute denotes an input or option field. If the translated attribute is zero, no attribute character should be written, otherwise output the translated attribute character:

```

TEST-I F-I NPUT-F IELD:
  TSTBUM(B)    ATTR, X' 80' /ONES(I NPUT-F IELD);
  CMPBLA(B)    ATTR, X' 00' /EQ(WRI TE-CONTENT);
  AND          . OUTBUF->DATA-STREAM(P: 1), ATTR, X' 3F' ;
  ADDN(S)      P, 1;

```

If the size of the item is greater than zero, copy the translated contents to the output buffer:

```

WRI TE-CONTENT:
  CMPNV(B)      F IELD-SI ZE, 0/EQ(=+2);
  XLATEWT      . OUTBUF->DATA-STREAM(P: F IELD-SI ZE),
               I TEM-DATA(1: F IELD-SI ZE), CHAR-TABLE; :

```

Finally, advance to the next item by adding the header size and the field size to the item pointer:

```

ADVANCE-TO-NEXT-I TEM:
  ADDN         LENGTH, I TEM-HEADER-SI ZE, F IELD-SI ZE;
  ADDSPP       . I TEM, . I TEM, LENGTH;
  ADDN(SB)     P, F IELD-SI ZE/POS(WRI TE-NEXT-I TEM);

```

For input fields save the position if no position has been saved yet, then for an error attribute reset it to the corresponding input attribute and save the position of the error-field if no error position has been saved yet:

```

I NPUT-F IELD:
  CMPBLA(B)    THE-POS I TION, "00000" /NEQ(=+2);
  CPYBLA       THE-POS I TION, I TEM-POS I TION; :

  VERI FY(B)   GOT, I TEM-ATTR, "EFGKefgk" /POS(WRI TE-SF-ORDER);
  XLATE        I TEM-ATTR, I TEM-ATTR, "EFGKefgk", "I JQRi j qr";
  CMPBLA(B)    ERR-POS I TION, "00000" /NEQ(=+2);
  CPYBLA       ERR-POS I TION, I TEM-POS I TION; :

```

## The VERI FY MI-Instruction

VERI FY      recei ver, source, cl ass;

We first met the VERI FY instruction in chapter 10. In the present chapter we use the instruction quite differently. To recapitulate: each character of the second operand (the *source*) character string value is checked to verify that it is among the (valid) characters indicated in the third operand (the *class*). If it is not among the valid characters, its character position in the source is stored in the first operand (the *receiver*) otherwise a zero value is stored in the receiver. As used here in our screen handler, where the source is a single character, a positive value of the receiver simply means that the source character was not present among the class characters (being the error attribute characters). A zero value means that the source character was an error attribute; we can then translate it to its corresponding input attribute, and store its buffer address position (if no error position was already stored).

## The XLATE MI-Instruction

XLATE      recei ver, source, compare, repl acement;

Each character of the second operand (the *source*) character string value is compared with the individual characters of the third operand (the *compare* operand). If a character of equal value does not exist in the compare operand, the source character is copied unchanged to the first operand (the *receiver*), otherwise the

character in the same relative position in the fourth operand (the *replacement*) as in the compare operand is copied in to the receiver. In our use of the XLATE instruction you can see how it's used to transpose 'E' to 'I', 'F' to 'J', etc.

We now construct the Field Format Word taking into account the bits for translating to upper case or rejecting input (for Option and Choice fields, hence the VERIFY against the "OC" class):

```
WRITE-SF-ORDER:
  CPYBLA      SF-FW, X' 4000' ;
  TSTBUM(B)   ATTR, X' 40' /ZER(=+2);
  OR(S)       SF-FW, X' 0020' ; /* TRANSLATE TO UPPER CASE */
  VERIFY(B)   GOT, ITEM-ATTR, "OC" /POS(=+2);
  OR(S)       SF-FW, X' 0600' ; /* REJECT INPUT, ONLY TAB */
```

We set the attribute character and the field length, output the completed Start Field (SF) order, update the output buffer position, and proceed to write the field contents (if any):

```
AND          SF-ATTR, ATTR, X' 3F' ;
CPYNV        SF-SIZE, FIELD-SIZE;
CPYBLA       .OUTBUF->DATA-STREAM(P: 6), SF-ORDER;
ADDN(SB)     P, 6/POS(WRITE-CONTENT);
```

### Calculating Where to Place the Cursor

If the cursor position was not given explicitly in the call control block, we determine where to put the cursor from the position of the first error field. If no fields are in error, we then try to use the position of the first input field. If there are no input fields defined, as the last resort, we position the cursor in the upper left-hand corner. Some manipulation may be needed to place the cursor on the first character within the field instead of on the attribute character:

```
SET-CURSOR:
  CMPBLA(B)   ERR-POSITION, "00000" /NEQ(USE-ERR-POSITION);
  CMPBLA(B)   CSR-POSITION, "00000" /NEQ(USE-CSR-POSITION);
USE-THE-POSITION:
  CPYBLA      ERR-POSITION, THE-POSITION; /* FIRST INPUT FIELD */
USE-ERR-POSITION:
  CPYNV(B)    BINARY-VALUE, ERR-ROW/ZER(NEXT-ROW); /* FIRST ERROR FIELD */
  CPYBLA      PUT-ROW, BINARY-CHAR;
  ADDN        BINARY-VALUE, ERR-COL, 1;
  CMPNV(B)    BINARY-VALUE, SCREEN-WIDTH/NHI (SEND-PUT-COMMAND);
NEXT-ROW:
  ADDLC(S)    PUT-ROW, X' 01' ;
  CPYNV(B)    BINARY-VALUE, 1/POS(SEND-PUT-COMMAND);
USE-CSR-POSITION:
  CPYNV        BINARY-VALUE, CURSOR-ROW; /* AS GIVEN IN CALL */
  CPYBLA      PUT-ROW, BINARY-CHAR;
  CPYNV        BINARY-VALUE, CURSOR-COL;
```

After setting the length of the send-buffer to include the cursor position we can finally send it off to be displayed:

```
SEND-PUT-COMMAND:
  CPYBLA      PUT-COL, BINARY-CHAR;
  ADDN        PUT-SEND-L, P, 8; /* CURSOR POS */
  CPYBLA      .OUTBUF->P-HEADER, PUT-HEADER;
  CALLX       .SEPT(PUT-FCT), PUT, *;
DONE:
  RTX        *;
```

### Read from Terminal

We ask for all the fields that have the modified data tag bit set (the MDT bit):

```
READ-TERMINAL:
  CPYBLA      .OUTBUF->IO-COMMAND, GET-MDT;
  CALLX       .SEPT(PUT-GET), GET, *;
  ADDSPP      INPUT, INBUF, 3;
```

The data returned starts out with the cursor row, column, and the AID key byte:

```
GET-CURSOR-POSITION:
  CPYBLA      BINARY-CHAR, INBUF->ROW-POS;
  CPYNV        CURSOR-ROW, BINARY-VALUE;
```

```

CPYBLA      BIN-CHAR,  . INBUF->COL-POS;
CPYNV       CURSOR-COL,  BINARY-VALUE;

```

We translate the raw AID character into our “cleaned up” value:

```

GET-CMD-KEY:
  XLATEWT    BIN-CHAR,  . INBUF->AID-KEY,  CMD-KEY-TABLE;
  CPYNV      CMD-KEY,  BINARY-VALUE;

```

The number of bytes in the input datastream is given by the ACTUAL-REC-SIZE variable in the I/O Feedback area. If this number is not higher than three we are done, as no data was modified and returned.

### Format of MDT Input Fields Returned

If any data is returned at all you get *all* the modified input fields. Leading and embedded nulls are converted to spaces while trailing nulls are stripped off. Each field is sent as an SBA-order:

SBA row column *data*

Note that no length information is present. Note also that the row/column information pertains to the first character of the *data*, not of any attribute that might precede the field. This means that when we compared the position with the positions of the screen items we must adjust for this difference.

To facilitate scanning of the input stream, we’ll place a stop-sentinel at the end of the input stream. The sentinel will be an SBA character followed by a row/column-value of binary zeroes:

```

DISTRIBUTE-MDT-DATA-IF-ANY:
  CMPNV(B)    ACTUAL-REC-SIZE,  3/NHI(DONE);
  ADDN        LENGTH,  ACTUAL-REC-SIZE,  1;
  CPYBLA      . INBUF->DATA-RETURNED(LENGTH: 3),  X'110000'; /* STOP */

```

Now, compute the adjusted field position and scan the screen items to find the field to receive the input data; if the position has a row value of zero we know that we are done because we have hit our sentinel:

```

GET-MDT-FIELDS:
  SETSPP      . ITEM,  SCREEN;
POSITION-TO-FIELD:
  CPYNV       BINARY-VALUE,  0;
  CPYBLA      BIN-CHAR,  INPUT(2:1); /* ROW */
  CPYNV(B)    THE-ROW,  BINARY-VALUE/ZER(DONE); /* FOUND STOP */
  CPYBLA      BIN-CHAR,  INPUT(3:1); /* COL */
  SUBN(B)     THE-COL,  BINARY-VALUE,  1/NZER(=+3);
  CPYNV       THE-COL,  SCREEN-WIDTH; /* LAST COLUMN IN */
  SUBN(S)     THE-ROW,  1; /* PREVIOUS ROW */

CHECK-POSITION:
  CMPBLA(B)   ITEM-ATTR,  " " /EQ(GET-MDT-FIELDS);
  CMPBLA(B)   THE-POSITION,  ITEM-POSITION/EQ(FOUND-MDT-FIELD);
  ADDN        LENGTH,  ITEM-HEADER-SIZE,  ITEM-SIZE;
  ADDSPP      . ITEM,  . ITEM,  LENGTH;
  B          CHECK-POSITION;

```

Otherwise we can now copy the data returned to the screen item. Because trailing blanks (nulls) are stripped off (even if they were actually entered by the operator) we have to provide these ourselves. To this end we need to know the length of the data returned for the field. Scanning for the SBA character introducing the *following* field gives us the needed information; we can then copy the data we got padding it with blanks to the end of the field. Special treatment must be given to the case where all data was stripped off because it was all blanks:

```

FOUND-MDT-FIELD:
  ADDSPP      . INPUT,  . INPUT,  3;
  CPYNV       FIELD-SIZE,  ITEM-SIZE;
  SCAN(B)     GOT,  INPUT,  X'11' /ZER(DONE);
  SUBN(SB)    GOT,  1/ZER(=+2);
  CPYBLAP     ITEM-DATA(1: FIELD-SIZE),  INPUT(1: GOT),  " ";
  CMPNV(B)    GOT,  0/HI(=+2);
  CPYBLAP     ITEM-DATA(1: FIELD-SIZE),  " ",  " ";

```

We then continue with the following field:

```

FIND-NEXT-FIELD:
  ADDSPP      . INPUT,  . INPUT,  GOT;
  B          POSITION-TO-FIELD;

```

## Finding Field With Cursor

The **MI TSTSCX** testprogram shows how to find the field where the operator left the cursor. An extra field in line 22 will contain the value of the **FIELD-FOUND** variable, the start and end of the field, and the cursor position. FIELD-FOUND will be zero if the cursor was not placed in a screen item. Here is the additional code that implements this functionality:

```
DCL DD * CHAR(10) INIT("L220010030");
DCL DD FOUND-AT CHAR(30) INIT(" ");

...
CALLI      FIND-FIELD, *, . FIND-FIELD;
CPYBLA     FOUND-AT( 1: 5), FIELD-FOUND;
CPYBLA     FOUND-AT( 7: 5), FIELD-START;
CPYBLA     FOUND-AT(13: 5), FIELD-END;
CPYBLA     FOUND-AT(20: 5), CTRL-CURSOR-POSITION;
B          SHOW-SCREEN;

...
DCL CON SCREEN-WIDTH      BIN(2) INIT(80);
DCL CON FIELD-HEADER-SIZE BIN(2) INIT(10);
DCL DD FIELD-ITEM-SIZE    BIN(2);
DCL MSPTR .FIELD;
DCL DD FIELD CHAR(3574)   BAS(.FIELD);
DCL DD FIELD-ATTR        CHAR(1) DEF(FIELD) POS( 1);
DCL DD FIELD-ROW         ZND(2,0) DEF(FIELD) POS( 2);
DCL DD FIELD-COL         ZND(3,0) DEF(FIELD) POS( 4);
DCL DD FIELD-SIZE        ZND(4,0) DEF(FIELD) POS( 7);
DCL DD FIELD-POSITION    CHAR(5) DEF(FIELD) POS( 2); /* REDEF */

DCL DD FIELD-FOUND CHAR(5);
DCL DD FIELD-START CHAR(5);
DCL DD START-ROW ZND(2,0) DEF(FIELD-START) POS(1);
DCL DD START-COL ZND(3,0) DEF(FIELD-START) POS(3);

DCL DD FIELD-END CHAR(5);
DCL DD END-ROW ZND(2,0) DEF(FIELD-END) POS(1);
DCL DD END-COL ZND(3,0) DEF(FIELD-END) POS(3);

DCL INSPTR .FIND-FIELD;
ENTRY      FIND-FIELD INT;
CPYBREP     FIELD-FOUND, "0";
SETSPP      FIELD, SCREEN;
NEXT-FIELD:
CMPBLA(B)   FIELD-ATTR, ". "/EQ(.FIND-FIELD);

CPYBLA      FIELD-START, FIELD-POSITION;
ADDN(S)     START-COL, 1;
: CMPNV(B)  START-COL, SCREEN-WIDTH/NHI(=+3);
SUBN(S)     START-COL, SCREEN-WIDTH;
ADDN(SB)    START-ROW, 1/POS(=-2);

CPYBLA      FIELD-END, FIELD-POSITION;
ADDN(S)     END-COL, FIELD-SIZE;
: CMPNV(B)  END-COL, SCREEN-WIDTH/NHI(=+3);
SUBN(S)     END-COL, SCREEN-WIDTH;
ADDN(SB)    END-ROW, 1/POS(=-2);

CMPBLA(B)    CTRL-CURSOR-POSITION, FIELD-START/LO(ADVANCE-TO-NEXT);
CMPBLA(B)    CTRL-CURSOR-POSITION, FIELD-END /HI(ADVANCE-TO-NEXT);
FOUND-FIELD-WITH-CURSOR:
CPYBLA      FIELD-FOUND, FIELD-POSITION;
B          .FIND-FIELD;

ADVANCE-TO-NEXT:
ADDN        FIELD-ITEM-SIZE, FIELD-HEADER-SIZE, FIELD-SIZE;
ADDSPP      FIELD, .FIELD, FIELD-ITEM-SIZE;
B          NEXT-FIELD;
```

The Devil lurks in the details, but this exercise should provide you with enough information to build your own screen handler should the simple version we've been studying not provide enough functionality. But we have shown that we can make the whole 5250-datastream process much easier to work with, by having the screen handler mask the complexity of the underlying byte stream.

In a later chapter, we will revisit describing screens and screen attributes and we will take the abstraction one step further and show how to describe a screen layout in XML, then transform the XML in to one of many formats - the initial format we will transform the XML in to will, as mentioned at the start of this



chapter, be the MI data item declaration. We will also see how much freedom having a XML description will give us in targeting different display mechanisms, other than a 5250-datastream.

## Chapter 18

### (SCV) Supervisor Call Vectored

#### The SCV/RFSCV Instructions

For the AS/400 implementation IBM introduced two additional (and undocumented) instructions for the PowerPC to allow an efficient way for one SLIC routine to call another or for a user program to call a SLIC routine. These instructions are called “Supervisor Call Vectored” (**SCV**) and “Return From Supervisor Call Vectored” (**RFSCV**). The **SCV** instruction identifies one of 128 locations. A separate SLIC routine can begin at each of the 128 locations and must be precisely 32 bytes (8 RISC instructions) long and usually ends up calling a much larger routine. The precise way this works has varied from release to release. A block of code 128 \* 32 bytes long constitutes a ‘dispatch table’ for the **SCV** instruction. This table is stored at real address **8000000000 003000** or to be more precise: at the location associated with the SLIC module “.scv0” with nickname **#CFSCV**.

A routine that is called through the SCV dispatch table can use virtual addresses, so that SLIC routines can be written to be pageable. When the **SCV** instruction is executed, the MSR processor mode bit is set to supervisor mode ensuring that the routines are able to execute supervisor-level RISC instructions, such as **MFMSR** (Move From Machine State Register). The **SCV** instruction does not use the normal interrupt save/restore registers SRR0 and SRR1. Instead, it uses the standard LINK and COUNT registers. The **RFSCV** instruction provides the return operation for the **SCV**.

#### How Is the SCV Used?

Let’s disassemble the following little program:

```
DCL SYSPTR .SYSPTR;
DCL DD RESOLVE CHAR(34);
DCL DD RESOLVE-TYPE CHAR( 2) DEF(RESOLVE) POS( 1) INIT(X' 0201' );
DCL DD RESOLVE-NAME CHAR(30) DEF(RESOLVE) POS( 3);
DCL DD RESOLVE-AUTH CHAR( 2) DEF(RESOLVE) POS(33) INIT(X' 0000' );

CPYBLAP      RESOLVE-NAME, "MYPGM", " ";
RSLVSP      .SYSPTR, RESOLVE, *, *;
RTX          *;
```

The RSLVSP-instruction results in this code:

0014E4	000064	387E0040	ADDI 3, 30, 64	; R3 => .SYSPTR
0014E8	000068	389E0050	ADDI 4, 30, 80	; R4 => RESOLVE
0014EC	00006C	E8A08130	LD 5, 0X8130	; R5 = NULL pointer
0014F0	000070	39400025	ADDI 10, 0, 37	; R10 = 37
0014F4	000074	38C50000	ADDI 6, 5, 0	; R6 = R5 = NULL pointer
0014F8	000078	44000141	SCV 10	; System Call Vectored to 10

The vector with selector value **10** is by far the most common SCV call. Registers from R3 and up are used to pass parameters to the call. Register R10 has a special purpose, as it holds the number of the function to be executed, in this case 37.

#### The SCV Dispatch Code

From V4R4M0 and onwards, the dispatch code looks like this:

Address	8000000000	003000				
003000	4B003002	60000000	60000000	60000000	SCV 0,	address FFFFFFFF 003000
003010	60000000	60000000	60000000	60000000		
003020	4B003022	60000000	60000000	60000000	SCV 1,	address FFFFFFFF 003020
003030	60000000	60000000	60000000	60000000		
...						
0030A0	4B0030A2	60000000	60000000	60000000	SCV 5,	address FFFFFFFF 0030A0
0030B0	60000000	60000000	60000000	60000000		
0030C0	4B0030C2	60000000	60000000	60000000	SCV 6,	address FFFFFFFF 0030C0
0030D0	60000000	60000000	60000000	60000000		
0030E0	4B0030E2	60000000	60000000	60000000	SCV 7,	address FFFFFFFF 0030E0
0030F0	60000000	60000000	60000000	60000000		
003100	4B003102	60000000	60000000	60000000	SCV 8,	address FFFFFFFF 003100

```

003110 60000000 60000000 60000000 60000000
003120 4B003122 60000000 60000000 60000000 SCV 9, address FFFFFFFF 003120
003130 60000000 60000000 60000000 60000000
003140 4B003142 60000000 60000000 60000000 SCV 10, address FFFFFFFF 003140
003150 60000000 60000000 60000000 60000000
003160 4B003162 60000000 60000000 60000000 SCV 11, address FFFFFFFF 003160
003170 60000000 60000000 60000000 60000000
...
003FC0 4B003FC2 60000000 60000000 60000000 SCV 126, address FFFFFFFF 003FC0
003FD0 60000000 60000000 60000000 60000000
003FE0 4B003FE2 60000000 60000000 60000000 SCV 127, address FFFFFFFF 003FE0
003FF0 00000000 E3C2E3C2 FFFFFFFF 92F5B3A0

```

The dispatch code consists of branch instructions, e.g. for SCV 10:

```

Address 800000000 003140
003140 4B003142 BA FF003140 [. . â] branch to absolute addr FFFFFFFF 003140
003144 60000000 ORI 00 00 000000 [- ] NOPs
003148 60000000 ORI 00 00 000000 [- ] ...
00314C 60000000 ORI 00 00 000000 [- ]
003150 60000000 ORI 00 00 000000 [- ]
003154 60000000 ORI 00 00 000000 [- ]
003158 60000000 ORI 00 00 000000 [- ]

```

Here is the code that is dispatched for SCV 10 (prior to V4R4M0, this code was stored directly within the dispatch table itself rather than the branch instruction now found - it is not clear what is gained by changing the mechanism to the newer indirect method):

```

Address FFFFFFFF 003140
003140 7C0902A6 MFSPR 00 09 [ @ . w ] ; R0 = old MSR
003144 7D8000A6 MFMSR 12 [ . Ø w ] ; R12 = current MSR
003148 780C042C RLDIMI 12 00 00 48 [ . . . ] ; R12(48:63) = R0(48:63)
00314C 79AC6CAC RLDIMI 12 13 13 50 [ . Ø%Ø ] ; R12(50:50) = R13(63:63) [is 0]
003150 419E889A BCA 12 30 FFFF8898 [ . Åhª ] ; if CR[30] = 1, single-step
003154 3C40B161 ADDI S 02 00 -20127 [ . £/ ] ;
003158 4A76AC66 BA FE76AC64 [ ¢TDA ] ; branch to FFFFFFFE 76AC64
00315C 60000000 ORI 00 00 000000 [- ] ;

```

The MFSPR 0, 9 instruction stores the COUNT register (containing the MSR before the interrupt) in R0. The MFMSR 12 instruction loads the current Machine State Register into R12. The two RLDIMI instructions merit further scrutiny.

## RLDIMI Instruction

Rotate Left Double Immediate, then Masked Insert:

`RLDIMI Ra, Rs, shift, maskbit` ;

The contents of the 64-bit register *Rs* are rotated left the number of bits specified by *shift* operand. A mask is generated having 1 bits from bit *maskbit* through bit 63 – *shift* and 0 bits elsewhere. The rotated data is *inserted* into *Ra* under control of the generated mask. Note that RLDIMI can be used to insert an *n*-bit field, that is right-justified in *Rs*, into *Ra* starting at bit position *b*, by setting *shift* = 64 – (*b* + *n*) and *maskbit* = *b*. The contents of *Ra* outside of the area where the mask bits are ones are *not* changed. The *shift* and *maskbit* fields are *split* fields (see chapter 6). Let's dissect the examples from the above code:

```

780C042C RLDIMI 12, 00, 00, 48 ;
  7 8 0 C 0 4 2 C
0111 1000 0000 1100 0000 0100 0010 1100 ; as hex digits expanded into bits
011110 00000 01100 00000 110000 011 0 0 ; bits grouped into instruction fields
  30 RO R12 SH MB 3 s 0 ; Rs is R0, Ra is R12, SH = 0, MB = 48

79AC6CAC RLDIMI 12, 13, 13, 50 ;
  7 9 A C 6 C A C
0111 1001 1010 1100 0110 1100 1010 1100 ; as hex digits expanded into bits
011110 01101 01100 01101 10010 011 0 0 ; bits grouped into instruction fields
  30 R13 R12 SH MB 3 s 0 ; Rs is R13, Ra is R12, SH = 13, MB = 50

```

In both cases the receiving register is R12 (already loaded with the new MSR). The first mask is:

0000 0000 0000 FFFF ; maskbits = 48 through 63 (63 – 0)

So, the first RLDIMI extracts the low-order 16 bits of the old MSR (loaded into R0) and inserts them into the low-order 16 bits of the copy of the new MSR in R12.

The second mask is:

0000 0000 0000 2000 ; maskbits = 50 through 50 (63 - 13)

First rotate R13 13 bits to the left:

8000 0000 0000 0000 => 0000 0000 0000 1000

So, a zero bit is inserted into R12[bit 50]. Bit 50 of the MSR is the floating-point enable/disable bit. What we are doing here is building the new value of the MSR to be used upon return from the SCV. Floating-point operations are always disabled by the SLIC to prevent unnecessary save/restore of the 32 floating-point registers. Should the program then issue a floating-point instruction, the trap-routine will turn floating-point mode back on again.

The mechanism of the call/return mechanism becomes clear from the code for an unused SCV-entry, e.g. SCV 0:

```
003000 7C0902A6 MFSPR 00 09      [ @ · w ] ; get saved MSR from COUNT register
003004 7D8000A6 MFMSR 12        [ ' 0 w ] ; get current MSR
003008 780C042C RLDIMI 12 00 00 48 [ l · · ] ; use bits 48-63 of saved MSR
00300C 79AC6CAC RLDIMI 12 13 13 50 [ ~ 0 % 0 ] ; clear bit 50 (floating-point bit)
003010 618C4030 ORI 12 12 016432 [ / 0 · ] ; set problem mode, address translation
003014 7D8903A6 MTSPR 09 12      [ ' i · w ] ; move new MSR to COUNT register
003018 4C0000A4 RFSCV            [ < u ] ; return (restores MSR from COUNT)
```

## Tracing and Single-Step Mode

We now continue with our analysis of the dispatch code. The Condition Register is divided into eight fields CF0 through CF7. The higher-numbered fields are used by the SLIC as various flags rather than as condition bits:

Address FFFFFFFF 003140

```
003150 419E889A BCA 12 30 FFFF8898 [ · A h a ] ; if CR[30] = 1, single-step
003154 3C40B161 ADDIS 02 00 -20127 [ · E / ] ;
003158 4A76AC66 BA FE76AC64 [ t I D A ] ; branch to FFFFFFFF 76AC64
00315C 60000000 ORI 00 00 000000 [ - ] ; filler
```

“Branch Conditionally to Absolute address” branches if it is true (the ‘12’) that bit 30 of the Condition register is set. Looking at the code at absolute address FFFFFFFF FF8898 we see that:

```
Address FFFFFFFF FF8898
FF8898 7C0000A6 MFMSR 00        [ @ w ] ; get MSR
FF889C 60000400 ORI 00 00 001024 [ - · ] ; set single-step mode, bit 53
FF88A0 7C000164 MTMSRD 00 00 00   [ @ · A ] ; set MSR
FF88A4 4C00012C I SYNC            [ < · · ]
FF88A8 3C40B161 ADDIS 02 00 -20127 [ · E / ]
FF88AC 4A76AC66 BA FE76AC64 [ t I D A ]
```

In single-step mode an exception occurs after each instruction has been executed. This allows a trace of the instruction stream to be recorded. Single-step mode is enabled by setting bit 53 in the Machine State Register. In either case, if we go through the setting up for single step code (i.e. when bit 30 of the condition register is set) or jump over it (i.e. when bit 30 is not set), we will always end up at address FFFFFFFF 76AC64 with register R2 set to point to an area with common procedures.

## ADDIS Instruction

Register R2 is set using the **ADDIS**-instruction:

003154 3C40B161 ADDIS 02 00 -20127 ; R2 = FFFF FFFF B161 0000

```
3 C 4 0 B 1 6 1
0011 1100 0100 0000 1011 0001 0110 0001 ; as hex digits expanded into bits
001111 00010 00000 1011000101100001 ; bits grouped into instruction fields
15 R2 R0 B161 ; R2 = sign-extend B161 shift left 16
```

“Add Immediate Shifted” loads the sign-extended x‘B161’ shifted 16 bits to the left into the destination

register. No addition is actually performed because the source register is 0. Register R2 now contains **FFFFFFFFB1 610000**. The purpose of this is to build an address to “high memory”. The PowerPC shows its 32-bit roots again.

## SCV 10 Handler

SST tells us that the name of the SCV 10 handler is **.#cfm1 r**:

```

                Display LIC Module Found
Address . . . . . : FFFFFFFFE 76AC60
Name . . . . . : .#cfm1 r

```

The code (and we quote) starts with the usual housekeeping, saving registers and return addresses:

```

76AC60 3C40B161 ADDIS 02 00 -20127 [· E/] ; R2 = FFFFFFFFB1 610000
76AC64 FBC1FFF3 STMD 30(01) -16 [0A· 3] ; store R30-R31
76AC68 7C0802A6 MFSPR 00 08 [· w] ; get LINK register (i.e. return address)
76AC6C F8010028 STD 00(01) 40 [8· ·] ; store return address in stack frame
76AC70 F821FF41 STD 01(01) -192 [8· ·] ; allocate new stack frame (size = 192)
76AC74 F9810020 STD 12(01) 32 [9a· ·] ; save MSR to be restored

76AC78 3C004010 ADDIS 00 00 16400 [· ·] ; frame type = x'4010'
76AC7C F8010008 STD 00(01) 8 [8· ·] ;

```

For now, you can just skim (or ignore) the following two short sections of code:

```

76AC80 7D7142A6 MFSPR 11 273 [· Éaw] ; get SPRG1 (the machine stack)
76AC84 7FE802A6 MFSPR 31 08 [·Y· w] ; get return address (again)
76AC88 800B00B0 LWZ 00(11) 176 [Ø· ^] ; get word W from machine stack
76AC8C 7D9E6378 OR 30 12 12 [· ÉA] ; R30 now copy of MSR
76AC90 70000400 ANDI 00 00 001024 [Ø· ·] ; if W has x'400' bit set (normally not)
76AC94 40820135 BCL 04 02 +308 [b· ·] ; call 76ADC8

76AC98 40940020 BC 04 20 +32 [m· ·] ; if CR[20] = 1 (normally not) --
76AC9C FBE10080 STD 31(01) 128 [Ü· Ø] ; store return address
76ACA0 F9410088 STD 10(01) 136 [9· h] ; save R10 (function no.)
76ACA4 39410080 ADDI 10 01 128 [· · Ø] ;
76ACA8 61ABD212 ORI 11 13 053778 [·K· ·] ;
76ACAC 4B006193 BLA FF006190 [· /I] ; do some debug stuff (?)
76ACB0 3C40B161 ADDIS 02 00 -20127 [· E/] ;
76ACB4 E9410088 LD 10(01) 136 [Z· h] ; restore R10

76ACB8 E982A468 LD 12(02) -23448 [ZbuÇ] ; R12 = DW (FFFFFFFFB1 60A468) <--

```

Now comes something interesting; R12 is being set to point to a table of byte values:

```

Address FFFFFFFFB1 60A460
60A460 FFFFFFFF 80989000 base of function table
60A468 FFFFFFFF 80989DD8 base of blocked flag table
60A470 FFFFFFFF B17A9110

```

## The Blocked Instruction Flag Table

Below are the contents of the table addressed by R12. The table contains a flag byte for each function number that register R10 may contain. The possible byte values are x'80' if the instruction for that function number is a *blocked* MI-instruction, otherwise x'00' for non-blocked instructions:

```

Address FFFFFFFF80 989DD0 byte numbers
989DD0 00008000 00808080 ; 1 8
989DE0 80000080 80808080 ; 9 24
989DF0 00808000 00000080 80808080 00000000 ; 25 40 <- function 37
989E00 00000080 00008080 80808080 00008000 ; 41 56
989E10 00808000 00808000 80800080 00008080 ; 57 72
989E20 80000080 80808080 00800080 80808080 ; 73 88
989E30 80800000 80800000 00808000 80808080 ; 89 104
989E40 00808080 00000080 00808080 80000000 ; 105 120
989E50 80000000 00000000 80808080 80000000 ; 121 136
989E60 00008080 80808080 80000000 00808080 ; 137 152
989E70 80808000 80000000 80008000 00808000 ; 153 168
989E80 00000000 80008080 80800080 00800080 ; 169 184
989E90 80808000 80808080 00808080 80800000 ; 185 200
989EA0 80808000 80800080 00008080 80808080 ; 201 216
989EB0 80808080 80808080 80808000 80008080 ; 217 232
989EC0 80008080 80000000 00000000 00000000 ; 233 248
989ED0 00800000 80800000 00000000 00000080 ; 249 264
989EE0 80808080 80800000 00000000 80808080 ; 265 280
989EF0 80008000 80008080 80808000 00000000 ; 281 296

```

```

989F00 00000000 00008080 80808080 80800000 ; 297 312
989F10 00000000 00000000 00808080 80808000 ; 313 328
989F20 00000080 00000000 80800000 00808080 ; 329 344
989F30 80800080 00000000 00000000 00000000 ; 345 360
989F40 00800000 80000000 00000000 00000000 ; 361 376
989F50 80000000 00000000 00000000 00008000 ; 377 392
989F60 00000000 00000000 80008080 00000080 ; 393 408
989F70 00808000 00000000 80800000 80000000 ; 409 424
989F80 00000000 00000000 00008080 80800000 ; 425 440
989F90 8080 ; 441 442

```

For example, if register R10 contains the function number '3', the above table flags that the corresponding MI-instruction, which for function number '3' is VMXCRTPG, is a blocked MI-instruction. An attempt to execute a blocked instruction will result in an exception at security levels from 40 and up. Here is how we load the flag byte for the function requested (remember from our test program that R10 still contains the function number '37'):

```

76ACBC 396AFFFF ADDI 11 10 -1 [· · ·] ; subtract 1 from function number
76ACCO 7D6C58AE LBZX 11 12 11 ['%i p] ; and offset into blocked instr. table
76ACC4 79600620 RLDI CL 00 11 00 56 [· · ·] ; extract low-order byte (redundant)

```

Now, test the program's state and the blocked instruction flag byte:

```

76ACC8 73CB0080 ANDI . 11 30 000128 [È 0] ; if MSR state-bit = 0 (system state)
76ACCC 4182000C BC 12 02 +12 [· b ·] ; branch around to 76ACD8 (ok)
76ACD0 700C0080 ANDI . 12 00 000128 [ø ·] ; if blocked flag not = x'80'
76ACD4 40820050 BC 04 02 +80 [ b &] ; branch down to 76AD24 (error)

```

So, if we come here, either the program is executing in system state or the flag byte is x'00'. In either case, the corresponding instruction is *not* considered blocked and execution continues:

```

76ACD8 7D7142A6 MFSPR 11 273 [' Éâw] ; get SPRG1 (machine stack)
76ACDC 396B0550 ADDI 11 11 1360 [· · &] ; set up address to some variable on stack
76ACE0 39800000 ADDI 12 00 0 [· ø ·] ; (don't know what's going on here, but ...)
76ACE4 998B0018 STB 12(11) 24 [r» ·] ; set the var. byte to binary zeroes

```

We still have to select a module to jump to. R10 still contains the function number, so:

```

76ACE8 7D5E5378 OR 30 10 10 [· : èl] ; copy function no. to R30
76ACEC E982A460 LD 12(02) -23456 [Zbu-] ; R12 = base of function table
76ACF0 70000001 ANDI . 00 00 000001 [ø ·] ; get low-order bit of flag byte (=0)
76ACF4 794B1F24 RLDI CR 11 10 03 60 [· · ·] ; (shift by 3 is multiply by 8)
76ACF8 396BFFF8 ADDI 11 11 -8 [· · 8] ; offset = funct * 8 - 8 = 8 * (funct - 1)
76ACFC 7D6C582A LDX 11 12 11 [· %i ·] ; R11 = address at table(offset)
76AD00 4182003C BC 12 02 +60 [· b ·] ; normally: branch around to 76AD3C

```

If the blocked flag byte had a low-order bit of 1 (we know of none), we would not have branched and we would end up here:

```

76AD04 F9610080 STD 11(01) 128 [9/ ø] ; some unusual condition:
76AD08 FBC10088 STD 30(01) 136 [ÜA h]
76AD0C FBE10090 STD 31(01) 144 [Ü. °]
76AD10 39410080 ADDI 10 01 128 [· · ø]
76AD14 61AB8CA0 ORI 11 13 036000 [· / øµ]
76AD18 4B0065F3 BLA FF0065F0 [· Å3]
76AD1C 3C40B161 ADDI S 02 00 -20127 [· É/]
76AD20 48000028 B +40 [ç ·] ; branch to common exit at 76AD48

```

If we come here, the instruction is blocked:

```

76AD24 7C1E0378 OR 30 00 00 [· @ · l]
76AD28 60000000 ORI 00 00 000000 [-]
76AD2C 480000E5 BL +228 [ç V] 76AE10
76AD30 60000000 ORI 00 00 000000 [-]
76AD34 7FC0F378 OR 00 30 30 [{" 3}]
76AD38 4800003C B +60 [ç ·] 76AD74

```

Here is where we actually execute the function:

```

76AD3C 7D6903A6 MTSPR 09 11 [· Ñ · w] ; move function address to COUNT
76AD40 4E800421 BCCTRL 20 00 [· ø ·] ; call address in COUNT register
76AD44 3C40B161 ADDI S 02 00 -20127 [· É/] ;

```

At the common exit point we have the usual housekeeping:

```

76AD48 7FD142A6 MFSPR 30 273 [· "Jâw] ; common exit: get machine stack ptr
76AD4C 91BE00B8 STW 13(30) 184 [j ' ½] ;

```

76AD50	A17E00FC	LHZ	11(30)	252	[~ = Ü]	
76AD54	2C0B0002	CMPI	0 11	000002	[. . .]	
76AD58	40820010	BC	04 02	+16	[ b . ]	76AD68
76AD5C	60000000	ORI	00 00	000000	[ - ]	
76AD60	4B013B8B	BLA		FF013B88	[. . . »]	
76AD64	3C40B161	ADDI S	02 00	-20127	[. £ /]	
76AD68	801E00B0	LWZ	00(30)	176	[Ø. ^]	
76AD6C	70000400	ANDI .	00 00	001024	[Ø . ]	
76AD70	40820059	BCL	04 02	+88	[ b β ]	76ADC8
76AD74	409E001C	BC	04 30	+28	[ Æ . ]	76AD90
76AD78	38C00400	ADDI	06 00	1024	[. { . ]	
76AD7C	38000020	ADDI	00 00	32	[. . .]	
76AD80	7D810028	LWARX	12 01 00		[ ' a . ]	
76AD84	7D8C3378	OR	12 12 06		[ ' Ø. l ]	
76AD88	7D81012D	STWCX.	12 01 00		[ ' a . ]	
76AD8C	4082FFF4	BC	04 02	-12	[ b. 4 ]	76AD80

And finally the return from the SCV:

76AD90	E8010020	LD	00(01)	32	[Y. . ]	; get saved MSR
76AD94	60004030	ORI	00 00	016432	[ - . ]	; set problem state
76AD98	7C0903A6	MTSPR	09 00		[@. . w]	; move to COUNT
76AD9C	382100C0	ADDI	01 01	192	[. . { ]	; release stack frame
76ADA0	E8010028	LD	00(01)	40	[Y. . ]	; get return address
76ADA4	7C0803A6	MTSPR	08 00		[@. . w]	; move to LINK
76ADA8	EBC1FFF3	LMD	30(01)	-16	[ØA. 3]	; restore R30-R31
76ADAC	4C0000A4	RFSCV			[< u]	; return (via LINK, set MSR from COUNT)

One interesting, and perhaps serious, point to make is that the Blocked Flag table is *not* in write-protected memory and can thus be altered permanently by any system-state program. A list of all SCV 10 functions is given in Appendix E.

## Other SCV Calls

Here is a list of some other MI-instructions implemented as separate SCV calls, rather than a selectable function of SCV 10:

SCV 3	SETPSPFP/CMPPSPAD
SCV 5	Return from CALLX
SCV 7	CALLX
SCV 8	XCTL
SCV 13	MATMDATA
SCV 14	ALCHSS
SCV 15	FREHSS
SCV 16	REALCHSS

## What Did We Learn?

From taking a look at the SCV instruction, it has shown us that a high percentage of the MI-instructions are actually implemented as selectable sub-functions of SCV 10. Using SCV 10 as a selector for the MI-instruction being used, offers IBM a useful abstracted layer, away from the ‘real’ code implementation for each MI-instruction, allowing IBM great flexibility for changing the underlying code for the instruction without impacting the program using the MI-instruction. We have also seen that the only barrier restricting user-written programs from using the blocked MI-instructions is a small, easily modifiable table.

## Chapter 19

### Calculating Archimedes' Constant, $\pi$

#### *An Amazing Formula for $\pi$*

People have been calculating the value of the ratio of a circle's perimeter to its diameter for millennia. In the 3<sup>rd</sup> century B.C., Archimedes (287 B.C. - 212 B.C.) considered inscribed and circumscribed regular polygons of 96 sides and deduced that  $3^{10/71} < \pi < 3^{1/7}$ . Today hundreds of billions of digits of  $\pi$  are known. Yet, people keep on calculating  $\pi$ . No book about computing is complete without rehashing this subject. Even my old S/38 MI-assembler manual exhibits a program to calculate  $\pi$ . In keeping with that proud tradition we'll include one here as well. Tremendous progress has occurred in the 25 years since the S/38 appeared. The speed of our hardware has increased by a factor of many thousands. What is often less appreciated is that many algorithms have been improved by an even greater factor, the best algorithm for calculating  $\pi$  being no exception. The current record-holder is the following quartically (the number of correct digits *quadruples* with each iteration) convergent algorithm, which is related to Ramanujan's work on elliptic integrals:

$$\begin{aligned}\alpha_0 &= 6 - 4 \cdot 2^{1/2}, & z_0 &= 2^{1/2} - 1 \\ z_{n+1} &= (1 - (1 - z_n^4)^{1/4}) / (1 + (1 - z_n^4)^{1/4}) \\ \alpha_{n+1} &= (1 + z_{n+1})^4 \alpha_n - 2^{2n+3} z_{n+1} [1 + z_{n+1} + z_{n+1}^2] \\ 1/\alpha_n &\Rightarrow \pi \text{ as } n \Rightarrow \infty\end{aligned}$$

This algorithm is the basis for Kanada's recent record-breaking evaluation of  $\pi$  to over 200 billion digits. Much more information about this and other related algorithms for calculating  $\pi$  can be found at <http://www.mathsoft.com/asolve/constant/pi/pi.html>. The first 10,000 digits of  $\pi$  can be found at <http://www.lacim.ugam.ca/piDATA/pi.html>. Here are the first thirty-five digits: 3.14159 26535 89793 23846 26433 83279 50288...

We'll present two implementations, one using packed decimal numbers and one using double-precision floating-point numbers. As a final touch, we'll show the code from the old S/38 MI-assembler manual.

#### *Computing the Square Root*

The algorithm contains several places where a square root must be computed. Firstly in the initial values where  $2^{1/2}$  is needed, but more importantly in the calculation of  $z_{n+1}$  where the fourth root (the square root of the square root) is called for. The square root must be computed correctly. Any error here propagates into the remainder of the calculation and is not reduced during the iteration.

The square root of  $N$  is calculated using Newton's iterative method with six iterations:

$$\begin{aligned}s_0 &= 1 \quad (\text{initial guess}) \\ s_{n+1} &= (s_n + N/s_n)/2\end{aligned}$$

We use packed numbers with maximal precision (31 digits):

```
DCL DD N      PKD(31, 30);
DCL DD SQRT   PKD(31, 30); /* SQRT = square root of N */

DCL DD K BIN(2);
DCL DD WRK   PKD(31, 30);
DCL INSPTR .GET-SQUARE-ROOT;
ENTRY      GET-SQUARE-ROOT INT;
  CPYNV          SQRT, 1; /* initial guess */
  CPYNV          K, 6;   /* six iterations */
NEWTON-SQRT-ITERATION:
  DIV           WRK, N, SQRT; /* NO ROUNDING */
  ADDN(S)       SQRT, WRK;
  DIV(SR)       SQRT, 2; /* ROUNDING MATTERS */

```



```

SUBN(SB)      K, 1/HI (NEWTON-SQRT-ITERATION);
B             .GET-SQUARE-ROOT;

```

It is important for maximum accuracy that the division by 2 is performed with rounding.

### The *MI PI PKD* Program

We'll iterate through the algorithm thrice, although, as we shall see, two iterations already give us the best approximation that we are going to obtain with the accuracy chosen:

```

DCL DD P   PKD(31, 0); /* power of 2 */
DCL DD Y   PKD(31, 30);
DCL DD A   PKD(31, 30);
DCL DD B   PKD(31, 30);
DCL DD C   PKD(31, 30);

CPYNV      P, 4;
CPYNV      N, 2;
CALLI      GET-SQUARE-ROOT, *, .GET-SQUARE-ROOT;

SUBN       Y, SQRT, 1;
MULT       B, SQRT, 4;
SUBN       A, 6, B;

CPYNV      M, 3; /* iterate 3 times */
: CALLI    ITERATE-PI, *, .ITERATE-PI;
SUBN(SB)   M, 1/HI (=1);
RTX        *;

DCL DD M   BIN(2);
DCL DD PI  ZND(31, 30);
DCL INSPTR .ITERATE-PI;
ENTRY      ITERATE-PI INT;
MULT       B, Y, Y;
MULT(S)    B, B;
SUBN       N, 1, B;
CALLI      GET-SQUARE-ROOT, *, .GET-SQUARE-ROOT;
CPYNV      N, SQRT;
CALLI      GET-SQUARE-ROOT, *, .GET-SQUARE-ROOT;
CPYNV      B, SQRT;

SUBN       Y, 1, B;
ADDN       C, 1, B;
DIV(SR)    Y, C; /* ROUNDING IS IMPORTANT */

ADDN       B, 1, Y;
MULT       C, Y, Y;
ADDN(S)    C, B;
MULT(S)    C, Y;
MULT(S)    C, P;

MULT(S)    P, 4;
MULT(S)    B, B;
MULT(S)    B, B;
MULT(S)    A, B;

SUBN(S)    A, C;
SUBN(S)    A, C;

DIV(R)     PI, 1, A;
CPYBLAP    MSG-TEXT, PI, " ";
CALLI      SHOW-MESSAGE, *, .SHOW-MESSAGE;
B          .ITERATE-PI;

```

```
%I NCLUDE SHOWMSG
```

The results after each of the three iterations are:

```

3.14159 26462 13542 28214 93444 32024 ; 7 correct decimals
3.14159 26535 89793 23846 26433 83260 ; 28 correct decimals
3.14159 26535 89793 23846 26433 83260 ; no change

```

If our machine could have handled it (or if we had programmed our own arithmetical operations on very long numbers), the third iteration would have given us 112 correct decimals, the next iteration 448, ... This is a truly remarkable performance.

## The *MIPIFLT* Program

And now for the floating-point version. We can use the square root function directly supported by the CMF1-instruction:

```

DCL DD N      FLT(8);
DCL DD SQRT   FLT(8);

DCL DD P      FLT(8);
DCL DD Y      FLT(8);
DCL DD A      FLT(8);
DCL DD B      FLT(8);
DCL DD C      FLT(8);

      CPYNV      P, 4;
      CPYNV      N, 2;
      CMF1       SQRT, X' 0020' , N;

      SUBN       Y, SQRT, 1;
      MULT       B, SQRT, 4;
      SUBN       A, 6, B;

      CPYNV      M, 3;
:      CALLI      ITERATE-PI, *, . ITERATE-PI;
      SUBN(SB)    M, 1/HI (= -1);

      RTX        *;

DCL DD M      BIN(2);
DCL DD PI     FLT(8);
DCL INSPTR . ITERATE-PI;
ENTRY      ITERATE-PI INT;
      MULT       B, Y, Y;
      MULT(S)    B, B;
      SUBN       N, 1, B;
      CMF1       SQRT, X' 0020' , N;
      CMF1       B, X' 0020' , SQRT;

      SUBN       Y, 1, B;
      ADDN       C, 1, B;
      DIV(S)     Y, C;

      ADDN       B, 1, Y;
      MULT       C, Y, Y;
      ADDN(S)    C, B;
      MULT(S)    C, Y;
      MULT(S)    C, P;

      MULT(S)    P, 4;
      MULT(S)    B, B;
      MULT(S)    B, B;
      MULT(S)    A, B;

      SUBN(S)    A, C;
      SUBN(S)    A, C;

      DIV        PI, 1, A;
      CPYNV      QQ, PI;
      CPYBLAP    MSG-TEXT, QQ, " ";
      CALLI      SHOW-MESSAGE, *, . SHOW-MESSAGE;
      B          . ITERATE-PI;

DCL DD QQ ZND(31,30);
%I NCLUDE SHOWMSG

```

Apart from simply changing the data type from PKD to FLT, the only real difference is that we cannot use the rounding options with floating-point instructions. Here is the result:

```

3.14159 26462 13546 83555 40979 24614 ; 7 correct
3.14159 26535 89809 54729 87279 20861 ; only 12 correct
3.14159 26535 89809 54729 87279 20861 ; no change

```

Even though the algorithm is quartic (so we should get four times as many correct digits in each iteration) we only get 12 correct decimals in the second iteration because the accuracy of floating-point numbers is only about 13-14 decimal digits.

## The Original S/38 Program, *MI PI S38*

The MI-assembler program from page 62 of my old S/38 notes (manual is too big a word) used the following formula:

$$\pi/4 = \arctan(1/7) + 2 \arctan(1/3)$$

where  $\arctan(x)$  is calculated from the series expansion:

$$\arctan(x) = x - x^3/3 + x^5/5 - x^7/7 + x^9/9 \dots$$

Here is the program with its *original* orthography intact:

```
DCL DD X      FLT(8)  AUTO INI T(E' +1. 0E00' )
DCL DD Y      FLT(8)  AUTO INI T(E' 7' )
DCL DD XS     FLT(8)  AUTO
DCL DD A      FLT(8)  AUTO INI T(E' 1' )
DCL DD DENOM  FLT(8)  AUTO INI T(E' 1. 0' )
DCL DD SUM    FLT(8)  AUTO
;
;
DCL DD TEMP   FLT(8)
DCL DD FOUR   FLT(8)  INI T(E' 4. 0E0' )
DCL DD PI     FLT(8)
;
;
DIV(S) X, 3;          /* X = 1/3          */
DIV Y, 1, Y;          /* Y = 1/7          */
MULT XS, X, X;        /* XS = X**2        */
NEG(S) XS;
ADDN A, X, 0;

LOOP1:
MULT(S) X, XS ;
ADDN(S) DENOM, 2 ;
DIV TEMP, X, DENOM;
ADDN(S) A, TEMP;
CMPNV(B) DENOM, 25 /LO(L00P1), EQ(L00P1) ;

MULT XS, Y, Y ;
NEG(S) XS;
ADDN SUM, Y, 0;
ADDN DENOM, 0, 1;
LOOP2: /* LOOP TO CALCULATE ATN(1/7) */
MULT(S) Y, XS ;
ADDN(S) DENOM, 2 ;
DIV TEMP, Y, DENOM ;
ADDN(S) SUM, TEMP ;
CMPNV(B) DENOM, 25 /LO(L00P2), EQ(L00P2) ;
ADDN PI, A, A;
ADDN(S) PI, SUM;
MULT(S) PI, FOUR;

CPYNV ZZ, PI ;
CPYBLAP MSG-TEXT, ZZ, " ";
CALLI SHOW-MESSAGE, *, . SHOW-MESSAGE;
RTX *;

DCL DD ZZ ZND(31, 30);
%I NCLUDE SHOWMSG
```

with the result

**3.14159 26535 89789 56328 42846 68043**

correct to 13 decimals. If we have learnt nothing else in the intervening 17 years it would be to appreciate the usefulness (indeed, necessity) of a neat program layout:

“Ugly programs are like ugly suspension bridges: they're much more liable to collapse than pretty ones, because the way humans perceive beauty is intimately related to our ability to process and understand complexity.”

- Eric S. Raymond

## Chapter 20

### Exception Handling

#### **Exceptions and Events**

At the hardware level, the processor deals with exceptions by issuing *interrupts*. An exception at this level is a condition (note how we keep hiding the real world behind yet another concept) that calls for changing the normal flow of instruction execution. At the MI-level, there is no concept of interrupts. MI distinguishes between *exceptions* and *events*. An exception at the MI is not the same as an exception at the hardware level (although the latter may cause the former), but is rather a ‘formally architected process message’ (F. Soltis).

An MI-exception is defined as either a machine-defined error detected during the execution of an instruction, or as a user-defined condition detected by a user-program. An event, on the other hand, is defined as an activity that occurs during machine operation that may be of interest to machine users. Exceptions are synchronous, meaning that they are caused by the execution of an instruction, while events are asynchronous, meaning that they are caused by actions outside the currently executing instruction. An example of a synchronous exception is an attempt to divide by zero. An example of an asynchronous event is the completion of an I/O operation.

Just as there are two types of exceptions (errors and user-defined conditions), there are also two types of events: object-related events and machine-related events. An MI-process can monitor the occurrence of a set of events and take appropriate action on some or all of them. Exceptions can also be monitored. Multiple monitors can be enabled at the same time. Each monitor has a priority controlling exception searching and handling when more than one monitor is active. Associated with each monitor is an exception handling routine; exceptions are formatted as messages and sent to the appropriate queue space.

In this chapter we shall only consider exceptions. Events are dealt with in later chapters. Superficially however, there are many similarities between exceptions and events.

#### **Declaring an Exception Monitor**

An exception monitor is installed by declaring an *Exception Description*. The syntax is as follows:

```
DCL EXCM exception-name EXCID( exception-number ) → [ INT
                                                         BP
                                                         EXT ] [ (handler-name) ]
                                                         >
                                                         ;
                                                         [ IMD
                                                         IGN
                                                         SKP
                                                         RSG
                                                         DFR ]
                                                         [ CV(string) ]
```

where

Element	Range	Description
<i>exception-name</i>	MI-name or *	Name of exception description
<i>exception-number</i>	Unsigned integer from 0 to 65535	Exception identifier
<i>handler-name</i>	MI-name	Name of exception handler
<i>string</i>	1 to 32 bytes	Compare value

Handler type	Description
INT	Name refers to an internal entry point
BP	Name is label for branch point
EXT	Name refers to system pointer for external program or entry point

The *exception action* determines what the system does when the exception is encountered:

Handler action	Description
IMD	Control passes immediately to the specified exception handler (default)
IGN	Ignore the exception and continue processing
SKP	Skip the current description, continue search for another description
RSG	Re-signal the exception and continue to search for a description
DFR	Postpone (defer) handling, saving exception data for later

## The Exception Identifier

Two bytes combined into an unsigned integer serve to identify an exception. The first byte is a *group* number, while the second byte is a subtype within the group. As further qualification the exception may have a *compare value*. Although compare values may contain up to 32 characters, OS/400 only uses three letter combinations, such as “CPF” or “MCH”. As exceptions are ‘architected’ as messages, they are assigned a textual message identifier consisting of the three-letter compare value followed by a four-digit *hexadecimal* (see below, though) representation of the exception identifier, such as “**CPF3CF1**” or “**MCH1202**”. Exceptions signaled by the machine have an internal compare value consisting of four null characters; although these are translated into the three-letter (more readable) value “MCH”. By historical accident (bummer...), machine exception IDs are converted byte for byte into *decimal* value message IDs. This silly convention has caused much grief and gnashing of teeth. For example, the dreaded “decimal data error” message we all hate and know as MCH1202 is really MCH0C02, so declaring an exception description to monitor for “decimal data error” (MCH1202), “zero divide” (MCH1210) and “size too small for result” (MCH1211) would look like this:

```
DCL EXCM DATA-ERROR EXCID(H'0C02', H'0C0A', H'0C0B') INT(ERROR) IMD CV("MCH");
```

An exception description may monitor for an exception with a *generic* ID as follows:

```
H'0000' - Any exception ID results in a match.
H'gg00' - Any exception ID in group gg results in a match.
H'ggmn' - The exception ID must match exactly ggm in order for a match to occur.
```

## Searching for Exception Descriptions

When an exception occurs, the exception descriptions of the current invocation are searched in the sequence in which they were declared. If an exception ID in an exception description corresponds to the occurring exception, the corresponding compare values are checked. If the compare value *length* in the exception description is less than the compare value length of the exception occurring, the length of the compare value in the exception description is used for matching purposes. If it is greater, an automatic mismatch results. As machine exception compare values have a length of four bytes, the three-character compare values used by OS/400 just squeak by.

## Materialize an Exception Description

You can materialize a named exception description by using the **MATEXCPD** instruction. The format is:

```
MATEXCPD .Receiver, Exception-description, Materialization-option;
```

The *.Receiver* operand is a space pointer to the materialization template. The materialization option is a one-character data item with the value x'00' meaning full materialization and higher values meaning partial materialization to various degrees. Here is the layout of the full template:

```
DCL SPCPTR .EXCP-DESCR INIT(EXCP-DESCR);
DCL DD EXCP-DESCR CHAR(96) BDRY(16);
DCL DD EXCP-DESCR-PROV BIN(4) DEF(EXCP-DESCR) POS(1) INIT(96);
DCL DD EXCP-DESCR-AVAIL BIN(4) DEF(EXCP-DESCR) POS(5);
DCL DD EXCP-MAT CHAR(88) DEF(EXCP-DESCR) POS(9);
DCL DD EXCP-MAT-CONTROL CHAR(2) DEF(EXCP-MAT) POS( 1);
DCL DD EXCP-MAT-INSTR-NBR BIN(2) DEF(EXCP-MAT) POS( 3);
DCL DD EXCP-MAT-CMPVAL-SIZE BIN(2) DEF(EXCP-MAT) POS( 5);
DCL DD EXCP-MAT-CMPVAL CHAR(32) DEF(EXCP-MAT) POS( 7);
DCL DD EXCP-MAT-NBR-OF-IDS BIN(2) DEF(EXCP-MAT) POS(39);
DCL SYSPTR .EXCP-MAT-HND-PGM DEF(EXCP-MAT) POS(41);
DCL DD EXCP-MAT-ID(16) CHAR(2) DEF(EXCP-MAT) POS(57);
```

The last data item, EXCP-MAT-ID, is an array of exception IDs. It is here configured for 16 entries, but can have many more (albeit a rare occurrence). As is customary with templates, it starts with two binary numbers showing how many bytes are provided in the template and how many were actually materialized.

The control flags, EXCP-MAT-CONTROL, determine the further treatment of the exception as follows

Bits	Value	Meaning
0-2	000	Exception Handler Action: Ignore
	001	Exception Handler Action: Skip
	010	Exception Handler Action: Resignal
	100	Exception Handler Action: Defer
	101	Exception Handler Action: Pass control immediately
3	0	Return exception data
	1	Do not return exception data
4	0	Reserved, must be zero
5	0	User data not present
	1	User data present
6-7	00	Reserved, must be zeroes
8-9	00	External entry point handler
	01	Internal entry point handler
	10	Branch point handler
10-15	000000	Reserved, must be zeroes

The instruction number, EXCP-MAT-INSTR-NBR, is the MI-instruction number to be given control when this exception occurs. If the exception handler is external, EXCP-MAT-INSTR-NBR will be set to zero (as it is not to be used). The size and contents of the compare value, EXCP-MAT-CMPVAL-SIZE and EXCP-MAT-CMPVAL, have their obvious meanings.

## Modifying an Exception Description

You can *modify* an exception description to alter the action to be taken. The simple form of the “Modify Exception Description” (**MODEXCPD**) instruction takes this format:

```
MODEXCPD    Exception-description, New-Control-Flags, X'01';
```

allowing you to specify a new set of control flags. You can also use a space pointer to an altered materialization template as operand 2 to change the exception description. You often modify an exception description after determining that the error causing the exception is indeed fatal and that no further repair of the situation makes sense or is even possible. In this case you should disable the exception monitor and retry the instruction (see below how to) causing the exception to occur again, but this time, due to your exception monitor having been disabled, it will cause the exception to be sent to the caller of your program. Here is how to disable the exception monitor:

```
MODEXCPD    Exception-description, X'2000', X'01';
```

## Monitoring Exceptions: The **MIDCEXC** Program

To illustrate some of the features of exception handling, we'll write a program to repair various decimal data errors. Whether such errors should be ignored, repaired, reported, or cause the program to fail is completely application dependent. We are, of course, not advocating that errors be repaired in all cases. What our particular example program will do is:

- In case of invalid decimal data (MCH1202), set the data to zeroes
- In case of divide by zero (MCH1210), set the data to the highest value possible (“infinity”)
- In case of result too large (MCH1211), set the data to the highest value possible (“infinity”)

For illustration, the program starts by materializing an exception description. Debug the program with a break point set at “1” to see the result (EXCP-DESCR). The program then continues to force the various errors, accumulating the result of each operation in a message that is issued at the end of the program. You

can set a break point at “2” and look at (EXCP-INFO) to see the exception data returned for each exception as it occurs.

```
DCL SPCPTR .EXCP-DESCR INIT(EXCP-DESCR);
DCL DD EXCP-DESCR CHAR(96) BDRY(16);
DCL DD EXCP-DESCR-PROV BIN(4) DEF(EXCP-DESCR) POS(1) INIT(96);
DCL DD EXCP-DESCR-AVAIL BIN(4) DEF(EXCP-DESCR) POS(5);
DCL DD EXCP-MAT CHAR(88) DEF(EXCP-DESCR) POS(9);
DCL DD EXCP-MAT-CONTROL CHAR(2) DEF(EXCP-MAT) POS( 1);
DCL DD EXCP-MAT-INSTR-NBR BIN(2) DEF(EXCP-MAT) POS( 3);
DCL DD EXCP-MAT-CMPVAL-SIZE BIN(2) DEF(EXCP-MAT) POS( 5);
DCL DD EXCP-MAT-CMPVAL CHAR(32) DEF(EXCP-MAT) POS( 7);
DCL DD EXCP-MAT-NBR-OF-IDS BIN(2) DEF(EXCP-MAT) POS(39);
DCL SYSPTR .EXCP-MAT-HND-PGM DEF(EXCP-MAT) POS(41);
DCL DD EXCP-MAT-ID(16) CHAR(2) DEF(EXCP-MAT) POS(57);

DCL EXCM DEC-ERROR EXCID(H'0C02', H'0C0A', H'0C0B')
                                INT(ERROR) IMD CV("MCH");
DCL DD DATA CHAR(3);
DCL DD NUMBER PKD(5,0) DEF(DATA) POS(1);

MATEXCPD .EXCP-DESCR, DEC-ERROR, X'00';
BRK "1"; /* TO SHOW DESCRIPTION */

CPYBREP DATA, " "; /* MAKE INVALID PACKED NUMBER */
CALLI SHOW-DATA, *, .SHOW-DATA;

ADDN(S) NUMBER, 1; /* FORCE 'DECIMAL DATA ERROR' */
CALLI SHOW-DATA, *, .SHOW-DATA;

DIV(S) NUMBER, 0; /* FORCE 'ZERO DIVIDE ERROR' */
CALLI SHOW-DATA, *, .SHOW-DATA;

ADDN(S) NUMBER, 1; /* FORCE 'NUMERIC SIZE ERROR' */
CALLI SHOW-DATA, *, .SHOW-DATA;

SUBN(S) NUMBER, 1; /* NO ERRORS, RESULT = 99998 */
CALLI SHOW-DATA, *, .SHOW-DATA;
CALLI SHOW-MESSAGE, *, .SHOW-MESSAGE;

RTX *;
```

## Exception General and Specific Data

When an exception occurs, general exception information about the location and nature of the exception can be retrieved. In addition, many exceptions also have specific data associated with them giving further details about the error or condition that has occurred. The MI Functional Reference Manual chapter 27 provides a lot of details about this specific data. The exceptions we are monitoring for in our test program happen not to return any specific data, but we'll structure the code to be prepared for specific data in order to make it easier to adapt the code for other exceptions. The “Retrieve Exception Data” (**RETEXCPD**) instruction with this format

```
RETEXCPD .Exception-info, Exception-handler-type;
```

retrieves the data related to the occurrence of the exception into the materialization area given by the space pointer *.Exception-info*. Here is the format of the information area:

```
DCL SPCPTR .EXCP-INFO INIT(EXCP-INFO);
DCL DD EXCP-INFO CHAR(304) BDRY(16);
DCL DD EXCP-INFO-PROV BIN(4) DEF(EXCP-INFO) POS( 1) INIT(304);
DCL DD EXCP-INFO-AVAIL BIN(4) DEF(EXCP-INFO) POS( 5);
DCL DD EXCP-INFO-ID CHAR(2) DEF(EXCP-INFO) POS( 9);
DCL DD EXCP-INFO-CMP-SIZE BIN(2) DEF(EXCP-INFO) POS(11);
DCL DD EXCP-INFO-CMPVAL CHAR(32) DEF(EXCP-INFO) POS(13);
DCL DD EXCP-INFO-REFKEY BIN(4) DEF(EXCP-INFO) POS(45);
DCL DD EXCP-DATA CHAR(256) DEF(EXCP-INFO) POS(49);
```

The exception data, EXCP-DATA, contains both variable-size exception *specific* data and a 46-byte fixed-size data part related to the invocation and location of where the exception occurred. Unfortunately, the variable-size data comes first. To access the fixed-part, it is thus convenient to declare a *based* structure:

```
DCL DD INVOCATION-PART BIN(2);
DCL SPCPTR .EXCP-INVOC;
DCL DD EXCP-INVOC CHAR(46) BAS(.EXCP-INVOC);
```

```

DCL PTR .EXCP-SOURCE-INVOC      DEF(EXCP-INVOC) POS( 1);
DCL PTR .EXCP-TARGET-INVOC      DEF(EXCP-INVOC) POS(17);
DCL DD EXCP-SOURCE-INSTR        BIN(2) DEF(EXCP-INVOC) POS(33);
DCL DD EXCP-TARGET-INSTR        BIN(2) DEF(EXCP-INVOC) POS(35);
DCL DD EXCP-MACHINE-DATA        CHAR(10) DEF(EXCP-INVOC) POS(37);

```

We have made room for 256 bytes of exception data. Compute the offset to the fixed part from the number of bytes available and add it to the space pointer to the exception information to get the basing space pointer to the invocation information:

```

SUBN      INVOCATION-PART, EXCP-INFO-AVAIL, 46;
ADDSPP    .EXCP-INVOC, .EXCP-INFO, INVOCATION-PART;

```

The source invocation, `.EXCP-SOURCE-INVOC`, identifies the invocation that caused the exception. The target invocation, `.EXCP-TARGET-INVOC`, identifies the invocation that is the target of the exception, i.e. the last invocation that was given the chance to handle the exception. For machine exceptions, this is the invocation incurring the exception. For user-signaled exceptions you may specify a different target. You also retrieve the number of the MI-instruction that caused the exception in the source invocation and the number of the MI-instruction that is currently being executed in the target invocation.

The second operand, *Exception-handler-type*, specifies the exception handler type as follows:

```

X'00'    retrieve for a branch point exception handler
X'01'    retrieve for an internal entry point exception handler
X'02'    retrieve for an external entry point exception handler

```

You normally begin exception handler processing with retrieving the exception data:

```

ENTRY ERROR INT;
RETEXCPD  .EXCP-INFO, X'01'; /* RETRIEVE FOR INTERNAL ENTRY */
BRK "2"; /* TO SHOW INFORMATION */

```

Now we implement the rules for handling the exceptions depending on the exception ID:

```

CMPBLA(B) EXCP-INFO-ID, X'0C02'/NEQ(=+2);
CPYNV(B)  NUMBER, 00000/ZER(=+2); /* DECIMAL ERROR */
CPYNV     NUMBER, 99999; /* OTHER ERRORS */

SUBN      INVOCATION-PART, EXCP-INFO-AVAIL, 46;
ADDSPP    .EXCP-INVOC, .EXCP-INFO, INVOCATION-PART;

```

Finally we want to return to the invocation with our “repaired” NUMBER:

```

CPYBWP    .EXCP-RTN-INVOC, .EXCP-SOURCE-INVOC;
RTNEXCP   .EXCP-RETURN;

```

## Return from Exception

When an external exception handler invocation or an internal exception handler subinvocation gets control and then has done what it needs to do to process the exception, you need to terminate the handler with the “Return from Exception” (**RTNEXCP**) instruction:

```
RTNEXCP    .Exception-return-template;
```

where the operand specifies a space pointer to a template that in turn will specify the instruction to return to, within a specified invocation. The RTNEXPD instruction cannot be executed (and is not needed) in a branch point internal exception handler (you just branch to where you want to go). The template specifies the return address as the invocation pointer `.EXCP-RTN-INVOC`:

```

DCL SPCPTR .EXCP-RETURN INIT(EXCP-RETURN);
DCL DD EXCP-RETURN CHAR(19) BDRY(16);
DCL PTR .EXCP-RTN-INVOC DEF(EXCP-RETURN) POS( 1);
DCL DD EXCP-MBZERO CHAR(1) DEF(EXCP-RETURN) POS(17) INIT(X'00');
DCL DD EXCP-ACTION CHAR(2) DEF(EXCP-RETURN) POS(18) INIT(X'0100');

```

The reserved variable `EXCP-MBZERO` must be binary zero. The action code, `EXCP-ACTION`, determines what happens next:



Code	Action
X'0200'	Resume execution with the instruction that follows the RTNEXCP instruction (terminating the internal exception handler subinvocation).
X'0100'	Resume execution with the instruction following the instruction causing the exception
X'0000'	Re-execute the instruction that caused the exception

In our example program we want to resume execution of the instruction that follows the instruction causing the exception, so we set the action code to x'0100'.

The following little routine just inserts the current value of our **NUMBER** into a message to show at the end of the program. A small thing to note is the **AUTO** attribute of the position variable, N. The default storage attribute is **STATIC**, meaning that the variable is only allocated storage once and that it keeps its value from invocation to invocation. With the **AUTO** attribute, the variable is re-allocated and re-initialized every time the program is called. In our example, that ensures that we start from the beginning of the message area every time:

```
DCL DD N AUTO BIN(2) INIT(1); /* reset every time PGM is run */
DCL INSPTR .SHOW-DATA;
ENTRY      SHOW-DATA INT;
  CMPNV(B)  N, 1/HI(=+2);
  CPYBREP   MSG-TEXT, " ";: /* clear to blanks if N = 1 */
  CVTHC     MSG-TEXT(N:6), DATA; /* contains NUMBER */
  ADDN(S)   N, 10;
  B         .SHOW-DATA;
```

```
%INCLUDE SHOWMSG
```

Running the program we get:

```
Type reply (if required), press Enter.
From . . . : LSVALLGAARD 12/06/00 13:28:24
404040 00000F 99999F 99999F 99998F
```

**NUMBER** starts out as the invalid packed decimal value x'404040' (blanks), then is repaired to zeroes, then set to the maximum value after the zero divide and after the attempt to add 1 more to it, and finally ends up being the valid value 9998.

## Signaling Exceptions: The **MISIGEXC** Program

In addition to monitoring for exceptions, you can also *signal* exceptions to occur. You can either *re-signal* the exception from inside an exception handler (in which case the invocation is known) or you can signal a *new* exception (we could use the word *condition* for such new exceptions). When you want to signal a condition, the first problem you have is to get an *invocation pointer* as the target for the exception. You use the "Materialize Invocation Attributes" (**MATINVAT**) instruction to get the invocation pointer.

### Materialize Invocation Attributes

The **MATINVAT** instruction causes either one specific attribute or a list of attributes of the designated invocation to be materialized. We'll only consider the first option here (as it gets rather complicated otherwise). The syntax is:

```
MATINVAT .Receiver, Invocation, .Selection-template;
```

Operand 1, *.Receiver*, specifies a space pointer to the area to receive the attribute. Operand 2, *Invocation*, identifies the source invocation whose attribute is to be retrieved. If this operand is the *null* operand, "\*", the invocation issuing the instruction is identified. Operand 3, *.Selection-template*, is a space pointer to a template that selects the attribute to be materialized. Most of the fields in the template have to do with how to store a list of attributes and can be set to zeroes if only one attribute is to be materialized:

```
DCL SPCPTR .MAT-RECEIVER INIT(MAT-RECEIVER);
DCL DD    MAT-RECEIVER CHAR(16) BDRY(16);
DCL PTR   .MAT-INVOC DEF(MAT-RECEIVER) POS(1);

DCL SPCPTR .MAT-SELECT INIT(MAT-SELECT);
DCL DD    MAT-SELECT CHAR(32);
DCL DD    MAT-NBR-ATTRS BIN(4) DEF(MAT-SELECT) POS( 1) INIT( 1);
```

```

DCL DD MAT-ATTR-FLAGS BIN(4) DEF(MAT-SELECT) POS( 5) INIT( 0);
DCL DD MAT-ATTR-OFFSET BIN(4) DEF(MAT-SELECT) POS( 9) INIT( 0);
DCL DD MAT-ATTR-STORED BIN(4) DEF(MAT-SELECT) POS(13) INIT( 0);

DCL DD MAT-ATTR-ID BIN(4) DEF(MAT-SELECT) POS(17) INIT( 1);
DCL DD MAT-RECV-FLAGS BIN(4) DEF(MAT-SELECT) POS(21) INIT( 0);
DCL DD MAT-RECV-OFFSET BIN(4) DEF(MAT-SELECT) POS(25) INIT( 0);
DCL DD MAT-RECV-LENGTH BIN(4) DEF(MAT-SELECT) POS(29) INIT(16);

```

The important fields are the number of attributes to materialize (NBR-ATTRs = 1), the attribute identifier (ATTR-ID = 1, for the invocation pointer), and the length of the receiver area (RECV-LENGTH = 16 bytes, because we're materializing a pointer, and that's how large and plump pointers are).

The program now starts by materializing the invocation pointer to its own invocation:

```

GET-OWN-INVOCATION:
    MATINVAT .MAT-RECEIVER, *, .MAT-SELECT;

```

## Signaling an Exception

The “Signal Exception” (**SIGEXCP**) instruction signals (“sends”) a new exception or re-signals an existing exception to the target invocation. The syntax is:

```

SIGEXCP .signal-info, .Exception-info;

```

The first operand is a space pointer to a template holding the target invocation pointer, .EXCP-TO-INVOC. Bits 0 of the EXCP-**OPTION** determines if we have a new exception (bit 0 = 0) or if we are re-signaling an existing exception (bit 0 = 1). Setting bit 2 to a 1 gives you control over which exception description to search first. We are happy with the default, which is simply the first:

```

DCL SPCPTR .EXCP-SIGNAL INIT(EXCP-SIGNAL);
DCL DD EXCP-SIGNAL CHAR(20) BDRY(16);
DCL PTR .EXCP-TO-INVOC DEF(EXCP-SIGNAL) POS( 1);
DCL DD EXCP-OPTION CHAR(1) DEF(EXCP-SIGNAL) POS(17) INIT(X'00');
DCL DD * CHAR(1) DEF(EXCP-SIGNAL) POS(18) INIT(X'00');
DCL DD EXCP-1ST-DESCR BIN(2) DEF(EXCP-SIGNAL) POS(19) INIT(1);

```

The second operand (which is ignored for a re-signal operation, as the data is already known) has the same basic format as the template used by the RTNEXCPD instruction:

```

DCL SPCPTR .EXCP-INFO INIT(EXCP-INFO);
DCL DD EXCP-INFO CHAR(64) BDRY(16);
DCL DD EXCP-BYTES-PROV BIN(4) DEF(EXCP-INFO) POS( 1) INIT(64);
DCL DD EXCP-BYTES-AVAIL BIN(4) DEF(EXCP-INFO) POS( 5);
DCL DD EXCP-ID BIN(2) DEF(EXCP-INFO) POS( 9);
DCL DD EXCP-CMP-SIZE BIN(2) DEF(EXCP-INFO) POS(11);
DCL DD EXCP-CMP-VAL CHAR(32) DEF(EXCP-INFO) POS(13);
DCL DD * BIN(4) DEF(EXCP-INFO) POS(45);
DCL DD EXCP-DATA CHAR(16) DEF(EXCP-INFO) POS(49);

```

Since we are not interested in the invocation data, we set the length provided for the information area, EXCP-BYTES-PROV, to only include the 16 bytes of exception specific data that our sample code will use.

The plan is now to signal two exceptions. The first one to be caught by our own exception monitor (or condition handler if you prefer), and the second one without a monitor so that the exception will be intercepted by the default exception handler that every process has. First the monitor for our very own condition “LSV777” (I just made this up):

```

DCL EXCM MY-CONDITION EXCID(H'7777') BP(GOT-CONDITION) IMD CV("LSV");

```

We now signal the condition:

```

SIGNAL-CONDITION:
    CPYBWP .EXCP-TO-INVOC, .MAT-INVOC;
    CPYBLA EXCP-ID, X'7777';
    CPYINV EXCP-CMP-SIZE, 3;
    CPYBLA EXCP-CMP-VAL, "LSV";
    CPYBLAP EXCP-DATA, "Exception Data", " ";
    SIGEXCP .EXCP-SIGNAL, .EXCP-DATA;

```

The condition handler is shown a little further on. After handling the condition, control returns to here, where we issue the data value error (MCH1223) exception:

```

SIGNAL-ERROR:
  CPYBLA      EXCP-ID,          X'0C17';      /* 1223 = Data value error */
  CPYNV       EXCP-CMP-SIZE,    4;             /* machine-generated compare length */
  CPYBLA      EXCP-CMP-VAL,     X'00000000';  /* MCH */
  SIGEXCP     .EXCP-SIGNAL, .EXCP-DATA;

```

Finally we return from the program:

```
RTX      *;
```

The condition handler is a branch point handler, so we must retrieve exception data as appropriate for a branch point:

```

GOT-CONDITION:
  RETEXCPD    .EXCP-INFO, X'00'; /* RETRIEVE FOR BRANCH POINT */
  CPYBLAP     MSG-TEXT, EXCP-DATA(1:16), " ";
  CALLI       SHOW-MESSAGE, *, .SHOW-MESSAGE;
  B           SIGNAL-ERROR;

```

```
%INCLUDE SHOWMSG
```

We simply show the data returned as a message:

```

Type reply (if required), press Enter.
From . . . : LSVALLGAARD    12/06/00    21:52:15
Exception Data

```

When the program finishes, the job log contains:

```

Data value error.
Processing command failure, refer to job log for details

```

## Various Errors

The default exception handler is the external program **QMHPDEH**. This program is rather picky. If you send it an exception it is not prepared to accept, you get the following error:

```

Message ID . . . . . : CPF2524      Severity . . . . . : 40
Message type . . . . . : Escape
Date sent . . . . . : 12/05/00      Time sent . . . . . : 22:46:22

```

```

Message . . . . . : Exception handler not available because of reason code 1.
Cause . . . . . : An error condition was found when an exception was sent.
The condition did not allow exception processing to complete normally. The
reason code within the message identifies the condition that was found.
Possible values for the reason code and their meanings are as follows:
Reason Code 1: an exception was signaled directly by using the SIGEXCP
instruction. The exception was not handled and QMHPDEH was called. The
exception message had compare data that did not allow QMHPDEH to convert
the exception into an MCH escape exception.

```

Or you may provoke a dump:

```

File . . . . . : QPSRVDMP                      Page/Line 1/1
Control . . . . . :                               Columns 1 - 78
Find . . . . . :
*...+...1...+...2...+...3...+...4...+...5...+...6...+...7...+...
5769SS1 V4R4M0 990521 AS/400 DUMP 265505/LSVALG
DUMP TAKEN FOR UNMONITORED ESCAPE MESSAGE
.MESSAGE ID- MCHFFFF
.MESSAGE FILE- QCPFMMSG LIBRARY- *LIBL
.SEVERITY- 00
.MSGTYPE- 0F
.SENDING-
..PROGRAM- #exsgexp
.LIBRARY-
..INSTRUCTION- 000000
.RECEIVING-
..PROGRAM- MISIGEXC LIBRARY- LSVALG
..INSTRUCTION- 0007
.MESSAGE-
MESSAGE NOT FOUND IN MESSAGE FILE
000000 C5E7C3C5 D7E3C9D6 D540C4C1 E3C14040
.POINTERS-
NONE
END OF DUMP

```

Or, even worse, cause an abnormal job termination:

Job 265505/LSVALGAARD/QPADEV0007 started on 12/06/00 at 10:05:36  
Incorrect exception ID was sent using the SIGEXCP instruction.  
Job ended abnormally.

Display the job log for more information.

```
MCH****   Escape      0J    12/06/00   10:22:27   #exsgexp           000000   MISIGEXC
LSVALGAARD
Message . . . . . : MCH**** QCPFMMSG   *LIBL
CPF240B   Information      40    12/06/00   10:22:27   QMHPDEH           QSYS   0455
*EXT      *N
Message . . . . . : Incorrect exception ID was sent using the SIGEXCP
Instruction.
Cause . . . . . : The exception ID of an unhandled exception message sent by
a program using the SIGEXCP instruction, was found to be incorrect. The
exception ID of this exception message was 8888. Recovery . . . : Change
the exception ID so its value is correct or report the problem to IBM.
CPC1219   Completion      50    12/06/00   10:22:55   QWTPITPP           QSYS   0585
*EXT      *N
Message . . . . . : This job ended abnormally.
Cause . . . . . : An error occurred that caused this job to end abnormally.
```

You get different errors depending on what the exception ID is, and what the compare value is. This situation is not really satisfactory. It is a research project of mine to find out why. You may want to join...

## Preventing Messages in the Joblog

When you run the **MIDCEXC** program the program catches all three exceptions and does its repair work as desired. There is, however, an undesirable (in most cases) side effect: For every exception caught, an entry is made in the job log for your job, *e.g.*:

```
3 > call midexcxc
Decimal data error.
Attempt made to divide by zero for fixed point operation.
Receiver value too small to hold result.
```

There could be thousands of such exceptions and although some indications of the occurrence of these exceptions would be desirable, it is better that the program itself issues a short summary at the end, rather than the system clogging up the job log with thousands of messages. So, the question is: Can we prevent an exception from generating a job log entry?

Since the *real* compare value generated by the machine is x'**00000000**' and not "**MCH**", we must monitor with *that* compare value, *i.e.*:

```
DCL EXCM DEC-ERROR EXCID(H'0C02', H'0C0A', H'0C0B')
INT(ERROR) IMD CV(X'00000000');
```

instead of (as we did):

```
DCL EXCM DEC-ERROR EXCID(H'0C02', H'0C0A', H'0C0B')
INT(ERROR) IMD CV("MCH");
```

If we do that, it turns out that no job log entry is generated. If we use CV("**MCH**"), the exception is really caught first by a lower layer monitoring for x'**00000000**'. That lower layer issues its joblog message and resignals a new exception (with compare value "**MCH**") for us to catch.

Blank

## Chapter 21

### Walking the ODP Chain

#### The Open Data Path

As we discussed in chapter 12, the Open Data Path (ODP) control block is the central control block of any open device or file. All ODPs for a job are *chained* together. By “walking” the chain you can get information about which files are currently being used (including file activity statistics). In chapter 9 we learned how to “walk” the Work Control Block Tables to look at every active job. From the Process Control Space of the job we can get access to the job’s ODPs. Combining the two “walks” opens up all kinds of interesting possibilities regarding gathering file activity information for your system in near real time. The present chapter shall be devoted to this topic.

#### Searching Work Control Block Tables

Borrowing logic and code from chapter 9, the basic loop through all active jobs is as shown below:

```
DCL DD SPCPTRS CHAR(48) BDRY(16);
  DCL SPCPTR .WCBSPC DEF(SPCPTRS) POS( 1);
  DCL SPCPTR .WCB-ROOT DEF(SPCPTRS) POS(17);
  DCL SPCPTR .WCB-ENTRY DEF(SPCPTRS) POS(33);

DCL DD OWN-PCO CHAR(512) BASPCO; /* our own PCO */
DCL SYSPTR @WCBT00 DEF(OWN-PCO) POS(433); /* points to WCB tables */

DCL DD WCB-ROOT CHAR(2048) BAS(.WCB-ROOT);
DCL SYSPTR .WCB-TABLES(30) DEF(WCB-ROOT) POS(577);

DCL DD THE-TABLE BIN(4);
DCL DD THE-OFFSET BIN(4);

DCL DD WCBTBL-SPACE CHAR(256) BAS(.WCBSPC);
DCL DD WCBTBL-SIZE BIN(4) DEF(WCBTBL-SPACE) POS(21);

DCL DD WCB-ENTRY CHAR(1024) BAS(.WCB-ENTRY);
DCL DD WCB-DEVICE CHAR(10) DEF(WCB-ENTRY) POS( 1);
DCL DD WCB-USER CHAR(10) DEF(WCB-ENTRY) POS( 11);
DCL DD WCB-NUMBER CHAR(6) DEF(WCB-ENTRY) POS( 21);
DCL SYSPTR .WCB-PCS DEF(WCB-ENTRY) POS( 33);
DCL DD WCB-TYPE CHAR(1) DEF(WCB-ENTRY) POS( 97);
DCL DD WCB-STATUS CHAR(1) DEF(WCB-ENTRY) POS( 98);
DCL DD WCB-GROUP CHAR(1) DEF(WCB-ENTRY) POS(100);
DCL DD WCB-START-TIME CHAR(8) DEF(WCB-ENTRY) POS(345);

GET-WCB-ROOT:
  SETSPFP .WCB-ROOT, @WCBT00; /* convert to space pointer */

SEARCH-WORK-CONTROL-TABLES:
  CPYV THE-TABLE, 0;
SEARCH-NEXT-TABLE:
  ADDN(S) THE-TABLE, 1;
  CMPV(B) THE-TABLE, 30 /HI (DONE-WITH-TABLES);
  CMPTRT(B) .WCB-TABLES(THE-TABLE), * /EQ(DONE-WITH-TABLES);
  SETSPFP .WCBSPC, .WCB-TABLES(THE-TABLE);

PREPARE-FOR-ENTRIES:
  SETSPFP .WCB-ENTRY, .WCBSPC;
  CPYV(B) THE-OFFSET, H' 0300' /POS(=+2); /* first entry */

NEXT-WCBTBL-ENTRY:
  ADDN(S) THE-OFFSET, H' 0400' :: /* size of each job entry */
  CMPV(B) THE-OFFSET, WCBTBL-SIZE /NLO(SEARCH-NEXT-TABLE);
  SETPPO .WCB-ENTRY, THE-OFFSET; /* point to entry */

CHECK-WCBTBL-ENTRY:
  CMPBLA(B) WCB-STATUS, X' 20' /NEQ (NEXT-WCBTBL-ENTRY); /*ACTIVE */

HAVE-ACTIVE-JOB-ENTRY:
  SETSPFP .WCB-PCO, .WCB-PCS;
  /* do something with this job entry */
  B NEXT-WCBTBL-ENTRY;
```

```
DONE-WITH-TABLES:
    RTX      *;
```

From the Process Communication Object (PCO) we pick up a system pointer to the “root” of the WCB tables. Then for each of up to 30 tables, we run through all jobs described in the table, looking for jobs that are actually running right now. Remember that most jobs in the WCB tables are really terminated jobs that still have spool files that have not been printed or otherwise released:

Review chapter 9 to make sure that the logic is clear in your mind. The code back in chapter 9 filtered out suspended jobs and non-running group jobs. We see no reason to be so picky here, so we shall include any job that is not terminated. OS/400 is architected with room for 30 WCB tables. No release does as yet use the full complement. Pointers to the tables are stored in the WCB “root”:

When we are done with scanning the tables, we are finished and can then either end processing and return or perform other similar housekeeping functions. We now have to specify what to do with each active job entry. We already have the pointer to the PCO for each job ‘extracted’ from the WCB tables, so we shall use this to locate the *Data Management Communication Object* (the so-called DMCQ) that contains the ODPs for that particular job

## The Data Management Communication Object

The DMCQ is a space object consisting of a *root* section and several *entries*. You locate the DMCQ as the third pointer in the PCO:

```
Address F406C12C9B 000020      /* sample PCO */
000020  80000000 00000000 2816099C 68001000  0· · · · · æÇ· · Space Ptr
000030  00008000 00000000 1B5CC0F2 4B00193F  · · 0· · · · · *(2· · Space
000040  00008000 00000000 FF91B163 890019FF  · · 0· · · · · j EÄi· · Space ← DMCQ
000050  00008000 00000000 1E63C896 DE0010FF  · · 0· · · · · AHou· · Dev Descr
000060  00008000 00000000 3555FD93 B60004FF  · · 0· · · · · i UI ¶· · Library
000070  80000000 00000000 17E686D9 61000100  0· · · · · WFR/· · Space Ptr
000080  80000000 00000000 E15A1E1A 5D000100  0· · · · · !· · )· · Space Ptr
```

A suitable MI-declaration would be:

```
DCL SPCPTR .WCB-PCO;
DCL DD WCB-PCO CHAR(1024) BAS(.WCB-PCO);
DCL SYSPTR @DMCQ DEF(WCB-PCO) POS( 33);
```

Here is what the system pointer points to:

```
Address FF91B16389 000000      /* DMCQ */
000000  00010008 00808000 FF91B163 89000000  · · · · 00· · j EÄi· ·
000010  C0010000 00000000 FF91B163 89000100  { · · · · · j EÄi· · Associated Space
000020  100019EF D8C4D4C3 D8404040 40404040  · · 0QDMCQ
000030  40404040 40404040 40404040 40404040
000040  40408000 00000F00 00000008 FF1C0301  0· · · · ·
000050  81B3D485 CF968000 00000000 00000000  a· Meö00· · · · ·
```

What we really want is the associated space at address **FF91B16389 000100** (that is where the data is stored). The “Set Space Pointer from Pointer” instruction does just that:

```
DCL SPCPTR .DMCQ;
SETSPFPF .DMCQ, @DMCQ; /* get space pointer */
```

## The DMCQ Root

The root section basically consists of a list of pointers to various objects involved with the DMCQ, such as the current Process Access Group, the Process Control Space, and the like, followed by a list of offsets to various *chains* of entries. One of these chains is the ODP chain. The chains have two-way links, and each chain is given by offsets to the last and first entry in the chain. Each entry then starts with offsets to the previous and to the next entry in the chain. A special chain is the *free* chain that contains entries currently not in use. If you open a file, the next free entry is assigned to the file and moved from the free chain to the ODP chain. When you close the file its entry is moved back onto the free chain. A chain ends when the next entry offset has the value -1. Because the chains ordinarily are bi-directional it doesn’t really matter

which of the two links we consider to be the forward link and which we consider to be the backward link. The free chain is a bit special as it has empty backward links (so is not really doubly-linked).

The entries are of several types (each type having its own chain):

Entry Type	Description
Free Entry	Space available for use
Open	Contains information about open files
Override	Information about overrides in effect
Override Parameters	Parameters from Override Command
Spool Override	Information about Spool overrides
Shared Mark Counts	Mark counters for shared ODPs
Device	Information about devices

We must explain what we mean by “an offset pointing to an entry”. To build a space pointer to an entry from the space pointer to the associated space and the offset to the entry, we use the “Set Space Pointer Offset” (**SETSPPO**) instruction.

### The SETSPPO MI-Instruction

The value of the binary number specified by operand 2 is assigned to the offset portion of the space pointer identified by operand 1. The space pointer continues to address the same space object:

SETSPPO            . *space-pointer*, *binary-offset*;

There is a subtle difference between the offset portion of a space pointer and the offset portion of an address. Take as an example the address of the associated space of the DMCQ we were just looking at: FF91B16389 000100. Its offset portion is x'000100'. If I wish to set the offset portion of the space pointer to that associated space to x'50', I should execute the instruction “SETSPPO . DMCQ, X' 50' ”. This sets the offset portion of the address to x'000150' because the space data portion already has an address offset of x'000100'; the offsets to use in the **SETSPPO** instruction are thus *relative* to the beginning of the associated space for the space object. It is clear that the space pointer must already point to an existing object.

Although we shall not go into details about the pointers stored in the root section, we'll show below the pointer section (for our sample DMCQ) anyway so that you have a general idea of what to expect. The data is taken from a “live” process with several open files:

```

000100 00008000 00000000 COEDEC23 6200013F ..0.... {00: A... Access Grp
000110 00008000 00000000 E87BD0F5 AD001A3F ..0.... Y#}5Y... Proc Ctrl
000120 AF000000 00000000 00000000 00000000 @.....
000130 80000000 00000000 0FA99A1E 95000100 0..... z^: n... Space Ptr
000140 00000000 00000000 00000000 00000000 .....
```

Here are some of the objects pointed to:

```

Address COEDEC2362 000000 /* process access group */
000000 00010010 00908000 COEDEC23 62000000 ..... °0: {00: A...
000010 00010000 00000000 00000000 FF000000 .....
000020 000001EF D7C1C740 40404040 40404040 ... 0PAG
000030 40404040 40404040 40404040 40404040
000040 40400000 00000000 00000010 FF1C0101 .....

Address E87BD0F5AD 000000 /* process control space */
000000 00010008 00800001 E87BD0F5 AD000000 ..... 0: Y#}5Y...
000010 00010000 00000000 F406C12C 9B000020 ..... 4: A: °...
000020 00001AEF D8D7C1C4 C5E5F0F0 F0D2D3E2 ... 0QPADEV000KLS
000030 E5C1D3C7 C1C1D9C4 F2F6F9F8 F5F04040 VALGAARD269850
000040 40408800 00000FE0 00003858 FF1C0101 h... \... i...

Address 0FA99A1E95 000000 /* data management entry point table */
000000 00010008 00808000 0FA99A1E 95000000 ..... 00: z^: n...
000010 C0010000 00000000 0FA99A1E 95000100 {..... z^: n...
000020 800019D7 D8C4D4C5 D7E3C240 40404040 0: PQDMEPTB
000030 40404040 40404040 40404040 40404040
000040 40400000 00000F00 00000008 3F100301 .....
```



The MI-declaration for the DMCQ begins to take shape:

```
DCL DD DMCQ CHAR(4096) BAS(. DMCQ);
DCL SYSPTR DMCQ-PROCESS-ACCESS-GROUP DEF(DMCQ) POS( 1);
DCL SYSPTR DMCQ-PROCESS-CONTROL-SPACE DEF(DMCQ) POS( 17);
DCL SPCPTR * DEF(DMCQ) POS( 33);
DCL SPCPTR DMCQ-DM-ENTRY-POINT-TABLE DEF(DMCQ) POS( 49);

DCL DD DMCQ-SIZE-OF-DMCQ BIN(4) DEF(DMCQ) POS(125);
DCL DD DMCQ-FIRST-FREE-ENTRY BIN(4) DEF(DMCQ) POS(129);
DCL DD DMCQ-LAST-FREE-ENTRY BIN(4) DEF(DMCQ) POS(133);
DCL DD * (4) BIN(4) DEF(DMCQ) POS(137);
DCL DD DMCQ-FIRST-SHARED-ODP BIN(4) DEF(DMCQ) POS(153);
DCL DD DMCQ-LAST-SHARED-ODP BIN(4) DEF(DMCQ) POS(157);
DCL DD DMCQ-FIRST-NON-SHARED-ODP BIN(4) DEF(DMCQ) POS(161);
DCL DD DMCQ-LAST-NON-SHARED-ODP BIN(4) DEF(DMCQ) POS(165);
```

Although there are many chains starting from the root, we are only really interested here in the ODP chains. For completeness we show here the entire chain starting-area (note that offsets to chains with no members are set to -1, showing up as hexadecimal values x'FFFFFFFF'):

000160	00000000	00000000	00000000	00000000	.....
000170	00000000	00000000	0000000B	00000F00	.....
000180	00000000	00000400	FFFFFFFF	FFFFFFFF	... \..... free chain
000190	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	..... shared ODP (empty)
0001A0	000003A0	00000220	00000100	FFFFFFFF	... μ..... non-shared ODP ←
0001B0	00000000	00000000	00000000	00000000	.....
0001C0	00000000	00000160	00000000	00000000	.....

We see that the offset to the first non-shared ODP is x'3A0'. Remembering that that translates into an address offset of x'4A0' we can show the entry for the first opened file (its ODP):

0004A0	00000340	FFFFFFFF	00000000	00000000	... ..
0004B0	80000000	00000000	D9B922C4	58008A00	Ø..... R% Di «. Space Ptr
0004C0	00000000	00000000	0533FFFF	FFFF0000	.....
0004D0	00008000	00000000	D3408A21	2E0019FF	.. Ø..... L «..... Space
0004E0	00000160	FFFFFFFF	00000340	00000000	.....
0004F0	00000004	00000000	00000000	00000000	.....

Address	D3408A212E	000000	/* file */ <-----
000000	00010020	00800001	D3408A21 2E000000
000010	00010000	00000000	D3408A21 2E000100
000020	100019FC	D3E2E2C3	D9C5C5D5 4040D3E2
000030	E5C1D3C7	C1C1D9C4	5CD54040 40404040
000040	40408000	00003F00	00000020 FF1C0301

We can now follow the **forward** chain from 3A0 through 340, 2E0, 280, 1C0, to 222. Note also the **backward** chain:

000440	000002E0	000003A0	00000000	00000000	... \..... μ.....
000450	80000000	00000000	C911D4AA	190044D0	Ø..... I· Mj... à} Space Ptr
000460	00000000	00000000	04CEFFFF	FFFF0000	..... Ø.....
000470	00008000	00000000	F2D4582A	33000DFF	.. Ø..... 2Mj..... Cursor
000480	00000160	000003A0	000002E0	00000000	..... μ..... \.....
000490	00000004	00000000	00000000	00000000	.....

Address	F2D4582A33	000000
000000	00010008	00810001
000010	00010000	00000000
000020	10000DEF	D8D4C9E2
000030	E5C1D3C7	C1C1D9C4
000040	40408000	000009B0

0003E0	00000280	00000340	00000000	00000000	... Ø.....
0003F0	80000000	00000000	C911D4AA	190046D0	Ø..... I· Mj... ā} Space Ptr
000400	00000000	00000000	0446FFFF	FFFF0000	..... ā.....
000410	00008000	00000000	FFBB8320	FA0019FF	.. Ø..... ]c· 3..... Space
000420	00000160	00000340	00000280	00000000	..... Ø.....
000430	00000004	00000000	00000000	00000000	.....

Address	FFBB8320FA	000000
000000	00010028	00800001
000010	00010000	00000000
000020	100019FC	D8C4E2E4
000030	C4C14040	40404040
000040	40408000	00004F00

000380	000001C0	000002E0	00000000	00000000	... {..... \.....
000390	80000000	00000000	DE3BB0FF	EC066CE0	Ø..... ü· ^· Ø· %\ Space Ptr

```

0003A0 00000000 00000000 0253FFFF FFFF0000 .....ë.....
0003B0 00008000 00000000 DD9E894E 130019FF ..ø.....ûÆi..... Space
0003C0 00000160 000002E0 000001C0 00000000 ..... \... {.....
0003D0 00000004 00000000 00000000 00000000 .....

Address DD9E894E13 000000
000000 00010020 00800001 DD9E894E 13000000 .....ø.....ûÆi.....
000010 00010000 00000000 DD9E894E 13000100 .....ûÆi.....
000020 100019FC D3E2E2C3 D9C5C5D5 4040D3E2 ...ULSSCREEN LS
000030 E5C1D3C7 C1C1D9C4 5CD54040 40404040 VALGAARD*N
000040 40408000 00003F00 00000020 FF1C0301 ø.....

0002C0 00000220 00000280 00000000 00000000 .....ø.....
0002D0 80000000 00000000 CCE1425D FA00E6A0 ø.....ö.ä)³.Wµ Space Ptr
0002E0 00000000 00000000 0130FFFF FFFF0000 .....
0002F0 00008000 00000000 E9AD5056 E40019FF ..ø.....ZŸ&i U... Space
000300 00000160 00000280 00000220 00000000 .....ø.....
000310 00000004 00000000 00000000 00000000 .....

Address E9AD5056E4 000000
000000 00010018 00800001 E9AD5056 E4000000 .....ø.....ZŸ&i U...
000010 00010000 00000000 E9AD5056 E4000100 .....ZŸ&i U...
000020 100019FC D8C4E4D6 C4E2D7C6 4040D8D7 ...UQDUODSPF OP
000030 C4C14040 40404040 5CD54040 40404040 DA *N
000040 40408000 00002F00 00000018 FF1C0301 ø.....

000320 FFFFFFFF 000001C0 00000000 00000000 ..... {.....
000330 80000000 00000000 E778B1A9 15001050 ø.....Xl Ez... & Space Ptr
000340 00000000 00000000 0000FFFF FFFF0000 .....
000350 00008000 00000000 C5B044F2 410019FF ..ø.....E^a2... Space
000360 00000160 000001C0 FFFFFFFF 00000000 ..... {.....
000370 00000004 00000000 00000000 00000000 .....

Address C5B044F241 000000 /* file */ <-----
000000 00010020 00800001 C5B044F2 41000000 .....ø.....E^a2...
000010 00010000 00000000 C5B044F2 41000100 .....E^a2...
000020 100019FC D8C4E4C9 F8F04040 4040D8E2 ...UQDUI 80 OS
000030 E8E24040 40404040 5CD54040 40404040 YS *N
000040 40408000 00003F00 00000020 FF1C0301 ø.....

000500 00000460 00000000 00000000 00000000 ...-..... First Free Entry
000510 00000000 00000000 00000000 00000000 .....
...
000F80 FFFFFFFF 00000000 00000000 00000000 ..... Last Free Entry
000F90 00000000 00000000 00000000 00000000 .....

000FE0 00000000 00000000 00000000 00000000 ..... Unused
000FF0 00000000 00000000 00000000 00000000 .....

```

There is some confusion in the code as to what is first and last, and next and previous, but because the links are bi-directional, it doesn't really matter.

## A DMCQ Open Entry

Each *open entry* has the format shown below. Besides the links to the previous/next entry, there are two starting points for chains of entries associated with the open entry. The most important information for our purpose is the system pointer, . DMCQ-ODP, to the ODP for the file:

```

DCL DD DMCQ-ENTRY CHAR(96) BAS(.WALK);
DCL DD DMCQ-NEXT-ENTRY-OFFSET BIN(4) DEF(DMCQ-ENTRY) POS( 1);
DCL DD DMCQ-PREV-ENTRY-OFFSET BIN(4) DEF(DMCQ-ENTRY) POS( 5);
DCL DD * CHAR(8) DEF(DMCQ-ENTRY) POS( 9);
DCL SPCPTR . DMCQ-WORK-SPACE DEF(DMCQ-ENTRY) POS(17);
DCL SPCPTR * DEF(DMCQ-ENTRY) POS(33);
DCL SYSPTR . DMCQ-ODP DEF(DMCQ-ENTRY) POS(49);
DCL DD DMCQ-NEXT-X BIN(4) DEF(DMCQ-ENTRY) POS(65);
DCL DD DMCQ-PREV-X BIN(4) DEF(DMCQ-ENTRY) POS(69);
DCL DD DMCQ-NEXT-Y BIN(4) DEF(DMCQ-ENTRY) POS(73);
DCL DD DMCQ-PREV-Y BIN(4) DEF(DMCQ-ENTRY) POS(77);
DCL DD DMCQ-FLAGS CHAR(4) DEF(DMCQ-ENTRY) POS(81);
DCL DD * CHAR(12) DEF(DMCQ-ENTRY) POS(85);

```

## Locating the ODPs

There are two chains of ODPs: a chain of shared ODPs and a chain of non-shared ODPs. We'll traverse ("walk") both chains. We start by setting up a variable (CHAIN) with the offset to the first shared ODP, then we walk that chain; next, we change the CHAIN variable to contain the offset to the first non-shared ODP

and walk that chain. As these offsets are 8 bytes apart, a simple loop incrementing CHAIN by 8 each time through the loop suffices; when the offset in the chain gets to be -1, we're at the end of the chain.

Here is the code to walk the chain(s). Note that we use a machine space pointer (MSPPTR) instead of a standard space pointer. This is more for illustration rather than for efficiency as the savings here are but slight:

```
DCL DD CHAIN BIN(4);
DCL MSPPTR .WALK;

GET-ODP:
  SETSPFPF .WALK, @DMCQ; /* to DMCQ root */
  CPYV .CHAIN, 148; /* ODP base */

GET-ODP-CHAIN:
  ADDN(S) .CHAIN, 8; /* next chain */
  CMPNV(B) .CHAIN, 164/HI (DONE-WITH-ODPS);
  SETSPP .WALK, .CHAIN;

WALK-ODP-CHAIN:
  CMPNV(B) DMCQ-NEXT-ENTRY-OFFSET, -1/EQ(GET-ODP-CHAIN);
  SETSPP .WALK, DMCQ-NEXT-ENTRY-OFFSET;
  SETSPFPF .ODP, .DMCQ-ODP; /* get space pointer to ODP */
  /* process the ODP */
  B WALK-ODP-CHAIN;

DONE-WITH-ODPS:
```

Note how the .WALK machine pointer is used to point to each open entry in turn. The pointer starts out pointing to the DMCQ root, is then modified to point to the beginning of a chain, and finally serves to point to each entry. This way of a pointer doing “triple duty” is convenient, but is strictly speaking not very good programming practice. What makes this practice somewhat palatable here is that the various manipulations of the pointer take place within a few lines of tight code.

We are finally at our goal: a space pointer, .ODP, to the ODP can be obtained from the system pointer to the ODP space object, .DMCQ-ODP, that is stored in the open-entry: SETSPFPF .ODP, .DMCQ-ODP.

## Structure of the ODP

The beginning of the ODP space contains three fixed-size sections. The first gives offsets to various other components stored in the ODP. The second section contains pointers to I/O buffers and areas, and the last section contains various status information such as the current record length, number of records, record number, etc. The complete layout is given as part of the **MI ODPWLK** program shown below.

Of particular interest are the so-called *feedback areas*. When you open a file, the *open feedback* is filled in with information about the file. After subsequent I/O-operations, the *I/O feedback* provides information about the status of the operation:

```
DCL DD ODP CHAR(1024) BAS(.ODP);
DCL DD ODP.OPEN-FEEDBACK BIN(4) DEF(ODP) POS(13);
DCL DD ODP.IO-FEEDBACK BIN(4) DEF(ODP) POS(21);
```

It is important to note that the I/O feedback area is not filled in after an open operation; only after a genuine I/O operation. You build space pointers to the feedback areas by adding the appropriate offsets to the pointer to the ODP:

```
ADDSP .OPEN-FEEDBACK, .ODP, ODP.OPEN-FEEDBACK; /* pointer to open feedback */
ADDSP .IO-FEEDBACK, .ODP, ODP.IO-FEEDBACK; /* pointer to I/O feedback */
```

## The Open Feedback

The complete layout of the Open Feedback area can be found in Appendix F. Of direct interest to us are the following fields:

```
DCL SPCPTR .OPEN-FEEDBACK;
DCL DD OPEN-FEEDBACK CHAR(256) BAS(.OPEN-FEEDBACK);
DCL DD ODP-TYPE CHAR(2) DEF(OPEN-FEEDBACK) POS( 1);
DCL DD ODP-FILE CHAR(10) DEF(OPEN-FEEDBACK) POS( 3);
```

```

DCL DD ODP-LIBRARY      CHAR(10) DEF(OPEN-FEEDBACK) POS( 13);
DCL DD ODP-MEMBER       CHAR(10) DEF(OPEN-FEEDBACK) POS( 49);
DCL DD ODP-MISC-FLAGS1  CHAR(1)  DEF(OPEN-FEEDBACK) POS(116);

```

The *ODPtype*, ODP-**TYPE**, distinguishes the three kinds of open data paths:

Type	ODP Type Description
"DS"	Display, tape, ICF, save, printer file not being spooled, or diskette file not being spooled
"DB"	Database member
"SP"	Printer or diskette file being spooled or inline data file

When we process the ODP, we may want to skip display files and other files of type "DS" as these files normally not grow in real time:

```

DCL INSPTR .PROCESS-THI S-ODP;
ENTRY      PROCESS-THI S-ODP INT;
ADDSP      .OPEN-FEEDBACK, .ODP, ODP.OPEN-FEEDBACK;

SKI P-I F-DI SPLAY-FI LE:
CMPBLA(B)  ODP-TYPE, "DS"/EQ(.PROCESS-THI S-ODP); /* return from processing */

```

Bit 4 in the miscellaneous flags item, ODP-**MISC-FLAGS1**, is set to *one* if the file is a logical file. We may wish to skip logical files as well:

```

SKI P-I F-LOGI CAL-FI LE:
TSTBUM(B)  ODP-MISC-FLAGS1, X' 08' /ONES(.PROCESS-THI S-ODP);

```

Remember, with the above line of code, that if bit 4 *is* set in ODP-**MISC-FLAGS1** the branch is not back to the PROCESS-THI S-ODP routine (and thus causing an infinite loop), but back to the 'caller' of PROCESS-THI S-ODP, the address of which is stored in the pointer variable .PROCESS-THI S-ODP, which in this case, returns control to the 'B WALK-ODP-CHAIN' line of code.

## The I/O Feedback

The complete layout of the I/O Feedback area can be found in Appendix F. The fields of interest are the current operation and the various I/O counts broken down per type of operation:

```

DCL SPCPTR .IO-FEEDBACK;
DCL DD IO-FEEDBACK CHAR(512) BAS(.IO-FEEDBACK);
DCL DD IO-WRITE-COUNT      BIN(4) DEF(IO-FEEDBACK) POS( 3);
DCL DD IO-READ-COUNT       BIN(4) DEF(IO-FEEDBACK) POS( 7);
DCL DD IO-UPDATE-COUNT     BIN(4) DEF(IO-FEEDBACK) POS(11);
DCL DD IO-OTHER-COUNT      BIN(4) DEF(IO-FEEDBACK) POS(15);
DCL DD IO-CUR-OPERATION    CHAR(1) DEF(IO-FEEDBACK) POS(20);

```

Although not documented, I've seen IO-CUR-OPERATION be X '00' in some rare cases; we want to exclude these as well:

```

HAVE-DATABASE-OR-SPOOLFI LE:
ADDSP      .IO-FEEDBACK, .ODP, ODP.IO-FEEDBACK;
CMPBLA(B)  IO-CUR-OPERATION, X' 00' / EQ(.PROCESS-THI S-ODP);

```

## Data to Collect about an Active File

We decide to collect the following data - exemplified by THE-ENTRY declaration:

```

DCL DD THE-ENTRY CHAR(64) BDRY(16);
DCL DD ENTRY-DEVICE      CHAR(10) DEF(THE-ENTRY) POS( 1); /* job stuff */
DCL DD ENTRY-USER        CHAR(10) DEF(THE-ENTRY) POS(11);
DCL DD ENTRY-NUMBER      CHAR(6)  DEF(THE-ENTRY) POS(21);
DCL DD ENTRY-TYPE        CHAR(1)  DEF(THE-ENTRY) POS(27);

DCL DD ENTRY-FI LE       CHAR(10) DEF(THE-ENTRY) POS(28); /* file stuff */
DCL DD ENTRY-LI BRARY    CHAR(10) DEF(THE-ENTRY) POS(38);
DCL DD ENTRY-OPERATI ON  CHAR(1)  DEF(THE-ENTRY) POS(48);
DCL DD ENTRY-COUNT       BIN(4)  DEF(THE-ENTRY) POS(49);
DCL DD ENTRY-MEMBER      CHAR(10) DEF(THE-ENTRY) POS(53);

```

The first section contains information about the job using the file (gotten from the WCB job entry) and the second section contains information about the current status of the file (gotten from the ODP). The ENTRY-

COUNT item is the combined count of all operations against the file. As we encounter active files we fill-in an entry for it in the above data-structure, then add the entry to a large array (for simplicity - you could also store the entries in a user space or even a file):

```
DCL DD CUR-ENTRY-NBR BIN(4);
DCL DD LAST-ENTRY-NBR BIN(4);
DCL DD ENTRIES(100000) CHAR(64) BDRY(16);

CPYBLA      ENTRY-OPERATION, IO-CUR-OPERATION;
CPYNV       ENTRY-COUNT,      IO-WRITE-COUNT;
ADDN(S)     ENTRY-COUNT,      IO-READ-COUNT;      /* add up all counts */
ADDN(S)     ENTRY-COUNT,      IO-UPDATE-COUNT;
ADDN(S)     ENTRY-COUNT,      IO-OTHER-COUNT;
CMPNV(B)    ENTRY-COUNT,      0/EQ(. PROCESS-THIS-ODP); /* if no activity: return */

CPYBLA      ENTRY-DEVICE,      WCB-DEVICE;
CPYBLA      ENTRY-USER,        WCB-USER;
CPYBLA      ENTRY-NUMBER,      WCB-NUMBER;
CPYBLA      ENTRY-TYPE,        WCB-TYPE;

CPYBLA      ENTRY-LIBRARY,     ODP-LIBRARY;
CPYBLA      ENTRY-FILE,        ODP-FILE;
CPYBLA      ENTRY-MEMBER,      ODP-MEMBER;

CMPNV(B)    LAST-ENTRY-NBR, 100000/NLO(. PROCESS-THIS-ODP);
ADDN(S)     LAST-ENTRY-NBR, 1; /* only if room left */
CPYBLA      ENTRIES(LAST-ENTRY-NBR), THE-ENTRY;
B           . PROCESS-THIS-ODP;
```

## The MI ODPWLK Program

Putting it all together we get the following program (**MI ODPWLK**). As it accesses system areas it has to have the system state attribute. Refer to chapter 7 for how to do this. To see the result (squirreled away in the ENTRIES array) set a breakpoint at statement "1":

====> **ADDBKP STMT(1) PGMVAR((ENTRIES ())) OUTFMT(\*HEX)**

The first time the program runs it may take a while to page all jobs into memory, but subsequent runs are usually very fast. You should see something like this:

```
Variable . . . . . : ENTRIES
Lower/upper bounds . . . . . : (1:100000)
Type . . . . . : CHARACTER
Length . . . . . : 64
Element * . . . . . 1 . . . . . * . . . . . 1 . . . . .
1 D8C4C2E2D9E5E7D94040D8E2E8E24040 'QDBSRVXR QSYS '
40404040F2F7F2F7F5F4E2D8C1C4C2C3 ' 272754SQADBC'
C3E2E34040D8E2E8E2404040404003 'CST QSYS '
00000039D8C1C4C2C3C3E2E340400000 'QADBCST '
2 D8C4C2E2D9E5E7D94040D8E2E8E24040 'QDBSRVXR QSYS '
40404040F2F7F2F7F5F4E2D8C1C4C2C6 ' 272754SQADBF'
C3E2E34040D8E2E8E2404040404005 'CST QSYS '
000000B5D8C1C4C2C6C3E2E340400000 'SQADBF CST '
etc...
```

We leave it as an exercise to turn that data into a useful report.

Here is then the complete **MI ODPWLK** program source:

```
DCL DD SPCPTRS CHAR(48) BDRY(16);
DCL SPCPTR .WCBSPC DEF(SPCPTRS) POS(1);
DCL SPCPTR .WCB-ROOT DEF(SPCPTRS) POS(17);
DCL SPCPTR .WCB-ENTRY DEF(SPCPTRS) POS(33);

DCL DD OWN-PCO CHAR(512) BASPCO;
DCL SYSPTR @WCBT00 DEF(OWN-PCO) POS(433);

DCL DD WCB-ROOT CHAR(2048) BAS(.WCB-ROOT);
DCL SYSPTR .WCB-TABLES(30) DEF(WCB-ROOT) POS(577);

DCL DD THE-TABLE BIN(4);
DCL DD THE-OFFSET BIN(4);

DCL DD WCBTBL-SPACE CHAR(256) BAS(.WCBSPC);
DCL DD WCBTBL-SIZE BIN(4) DEF(WCBTBL-SPACE) POS(21);
```

```

DCL DD WCB-ENTRY CHAR(1024) BAS(.WCB-ENTRY);
DCL DD WCB-DEVICE CHAR(10) DEF(WCB-ENTRY) POS( 1);
DCL DD WCB-USER CHAR(10) DEF(WCB-ENTRY) POS( 11);
DCL DD WCB-NUMBER CHAR(6) DEF(WCB-ENTRY) POS( 21);
DCL SYSPTR .WCB-PCS DEF(WCB-ENTRY) POS( 33);
DCL DD WCB-TYPE CHAR(1) DEF(WCB-ENTRY) POS( 97);
DCL DD WCB-STATUS CHAR(1) DEF(WCB-ENTRY) POS( 98);
DCL DD WCB-GROUP CHAR(1) DEF(WCB-ENTRY) POS(100);
DCL DD WCB-START-TIME CHAR(8) DEF(WCB-ENTRY) POS(345);

DCL SPCPTR .WCB-PCO;
DCL DD WCB-PCO CHAR(1024) BAS(.WCB-PCO);
DCL SYSPTR @DMCQ DEF(WCB-PCO) POS(33);

ENTRY * EXT;
GET-WCB-ROOT:
    SETSPFP .WCB-ROOT, @WCBTOO;

SEARCH-WORK-CONTROL-TABLES:
    CPYV LAST-ENTRY-NBR, 0;
    CPYV THE-TABLE, 0;
SEARCH-NEXT-TABLE:
    ADDN(S) THE-TABLE, 1;
    CMPNV(B) THE-TABLE, 30 /HI (DONE-WITH-TABLES);
    CMPTRT(B) .WCB-TABLES(THE-TABLE), */EQ(DONE-WITH-TABLES);
    SETSPFP .WCBSPC, .WCB-TABLES(THE-TABLE);

PREPARE-TO-SEARCH-WCB-TABLE:
    SETSPFP .WCB-ENTRY, .WCBSPC;
    CPYV(B) THE-OFFSET, H' 0300' /POS(=+2);

NEXT-WCBTBL-ENTRY:
    ADDN(S) THE-OFFSET, H' 0400' ;
    CMPNV(B) THE-OFFSET, WCBTBL-SIZE/NLO(SEARCH-NEXT-TABLE);
    SETSPP .WCB-ENTRY, THE-OFFSET;

CHECK-WCBTBL-ENTRY:
    CMPBLA(B) WCB-STATUS, X' 20' /NEQ (NEXT-WCBTBL-ENTRY); /*ACTIVE */

HAVE-ACTIVE-JOB-ENTRY:
    SETSPFP .WCB-PCO, .WCB-PCS;
    CALLI PROCESS-JOB-ENTRY, *, .PROCESS-JOB-ENTRY;
    B NEXT-WCBTBL-ENTRY;

DONE-WITH-TABLES:
    BRK "1"; /* BREAK POINT TO LOOK AT ENTRIES */
    RTX *;

/*****/

DCL DD CUR-ENTRY-NBR BIN(4);
DCL DD LAST-ENTRY-NBR BIN(4);
DCL DD ENTRIES(100000) CHAR(64) BDRY(16);

DCL DD THE-ENTRY CHAR(64) BDRY(16);
DCL DD ENTRY-DEVICE CHAR(10) DEF(THE-ENTRY) POS( 1);
DCL DD ENTRY-USER CHAR(10) DEF(THE-ENTRY) POS(11);
DCL DD ENTRY-NUMBER CHAR(6) DEF(THE-ENTRY) POS(21);
DCL DD ENTRY-TYPE CHAR(1) DEF(THE-ENTRY) POS(27);
DCL DD ENTRY-FILE CHAR(10) DEF(THE-ENTRY) POS(28);
DCL DD ENTRY-LIBRARY CHAR(10) DEF(THE-ENTRY) POS(38);
DCL DD ENTRY-OPERATION CHAR(1) DEF(THE-ENTRY) POS(48);
DCL DD ENTRY-COUNT BIN(4) DEF(THE-ENTRY) POS(49);
DCL DD ENTRY-MEMBER CHAR(10) DEF(THE-ENTRY) POS(53);

DCL SPCPTR .DMCQ;
DCL DD DMCQ CHAR(4096) BAS(.DMCQ);
DCL SYSPTR DMCQ-PROCESS-ACCESS-GROUP DEF(DMCQ) POS( 1);
DCL SYSPTR DMCQ-PROCESS-CONTROL-SPACE DEF(DMCQ) POS( 17);
DCL SPCPTR * DEF(DMCQ) POS( 33);
DCL SPCPTR DMCQ-DM-ENTRY-POINT-TABLE DEF(DMCQ) POS( 49);

DCL DD DMCQ-SIZE-OF-DMCQ BIN(4) DEF(DMCQ) POS(125);
DCL DD DMCQ-FIRST-FREE-ENTRY BIN(4) DEF(DMCQ) POS(129);
DCL DD DMCQ-LAST-FREE-ENTRY BIN(4) DEF(DMCQ) POS(133);
DCL DD *(4) BIN(4) DEF(DMCQ) POS(137);
DCL DD DMCQ-FIRST-SHARED-ODP BIN(4) DEF(DMCQ) POS(153);
DCL DD DMCQ-LAST-SHARED-ODP BIN(4) DEF(DMCQ) POS(157);
DCL DD DMCQ-FIRST-NON-SHARED-ODP BIN(4) DEF(DMCQ) POS(161);
DCL DD DMCQ-LAST-NON-SHARED-ODP BIN(4) DEF(DMCQ) POS(165);

```

```

DCL DD DMCQ-ENTRY CHAR(96) BAS(.WALK);
DCL DD      DMCQ-NEXT-ENTRY-OFFSET BIN(4) DEF(DMCQ-ENTRY) POS( 1);
DCL DD      DMCQ-PREV-ENTRY-OFFSET BIN(4) DEF(DMCQ-ENTRY) POS( 5);
DCL DD      *                      CHAR(8) DEF(DMCQ-ENTRY) POS( 9);
DCL SPCPTR . DMCQ-WORK-SPACE          DEF(DMCQ-ENTRY) POS(17);
DCL SPCPTR *                          DEF(DMCQ-ENTRY) POS(33);
DCL SYSPTR . DMCQ-ODP                 DEF(DMCQ-ENTRY) POS(49);
DCL DD      DMCQ-NEXT-X               BIN(4) DEF(DMCQ-ENTRY) POS(65);
DCL DD      DMCQ-PREV-X               BIN(4) DEF(DMCQ-ENTRY) POS(69);
DCL DD      DMCQ-NEXT-Y               BIN(4) DEF(DMCQ-ENTRY) POS(73);
DCL DD      DMCQ-PREV-Y               BIN(4) DEF(DMCQ-ENTRY) POS(77);
DCL DD      DMCQ-FLAGS                 CHAR(4) DEF(DMCQ-ENTRY) POS(81);
DCL DD      *                      CHAR(12) DEF(DMCQ-ENTRY) POS(85);

```

```

DCL DD CHAIN BIN(4);
DCL MSPPTR .WALK;

```

```

DCL INSPTR .PROCESS-JOB-ENTRY;
ENTRY      PROCESS-JOB-ENTRY INT;
CPYBLA     ENTRY-DEVICE, WCB-DEVICE;
CPYBLA     ENTRY-USER,   WCB-USER;
CPYBLA     ENTRY-NUMBER, WCB-NUMBER;
CPYBLA     ENTRY-TYPE,   WCB-TYPE;

```

```

GET-ODP:
SETSPFPF .WALK, @DMCQ; /* TO DMCQ ROOT */
CPYNV    CHAIN, 144; /* ODP BASE */

```

```

GET-ODP-CHAIN:
ADDN(S)    CHAIN, 8; /* NEXT CHAIN */
CMPNV(B)   CHAIN, 160/HI (DONE-WITH-ODPS);
SETSPPO    .WALK, CHAIN;

```

```

WALK-ODP-CHAIN:
CMPNV(B)    DMCQ-NEXT-ENTRY-OFFSET, -1/EQ(GET-ODP-CHAIN);
SETSPPO     .WALK, DMCQ-NEXT-ENTRY-OFFSET;
SETSPFPF    .ODP, .DMCQ-ODP; /* GET SPACE POINTER TO ODP */
CALLI       PROCESS-THIS-ODP, *, .PROCESS-THIS-ODP;
B           WALK-ODP-CHAIN;

```

```

DONE-WITH-ODPS:
B           .PROCESS-JOB-ENTRY;

```

```

/*****

```

```

DCL SPCPTR .ODP;
DCL DD ODP CHAR(1024) BAS(.ODP);
DCL DD ODP-STATUS      CHAR(4) DEF(ODP) POS( 1);
DCL DD ODP-DEV-LENGTH  BIN(4) DEF(ODP) POS( 5);
DCL DD ODP-OPEN-SIZE   BIN(4) DEF(ODP) POS( 9);
DCL DD ODP.OPEN-FEEDBACK BIN(4) DEF(ODP) POS(13);
DCL DD ODP.DEV-NAMELIST BIN(4) DEF(ODP) POS(17);
DCL DD ODP.IO-FEEDBACK  BIN(4) DEF(ODP) POS(21);
DCL DD ODP.LOCK-LIST    BIN(4) DEF(ODP) POS(25);
DCL DD ODP.SPOOL-OUTPUT BIN(4) DEF(ODP) POS(29);

DCL DD ODP.MBR-DESCR    BIN(4) DEF(ODP) POS(33);
DCL DD ODP.CUR-IN-REC   BIN(4) DEF(ODP) POS(37);
DCL DD ODP.CUR-OUT-REC  BIN(4) DEF(ODP) POS(41);
DCL DD ODP.OPEN-DMCQ    BIN(4) DEF(ODP) POS(45);
DCL DD ODP.OUTSTANDINGS BIN(4) DEF(ODP) POS(49);
DCL DD *                CHAR(12) DEF(ODP) POS(53);

DCL SYSPTR .ODP-CURSOR      DEF(ODP) POS( 65);
DCL SPCPTR *                DEF(ODP) POS( 81);
DCL SPCPTR .ODP-CDM-ERROR    DEF(ODP) POS( 97);
DCL SPCPTR .ODP-INPUT-BUFFER DEF(ODP) POS(113);
DCL SPCPTR .ODP-OUTPUT-BUFFER DEF(ODP) POS(129);

DCL DD ODP.CDM-CLOSING  BIN(2) DEF(ODP) POS(145);
DCL DD ODP-DEV-NAME-IDX BIN(2) DEF(ODP) POS(147);
DCL DD ODP-NBR-OF-DEVS  BIN(2) DEF(ODP) POS(149);

DCL DD ODP-SEQUENCE-NBR BIN(4) DEF(ODP) POS(151);
DCL DD ODP-REC-LENGTH   BIN(2) DEF(ODP) POS(155);
DCL DD ODP-REC-LENGTH2  BIN(2) DEF(ODP) POS(157);
DCL DD ODP-NBR-OF-*RDS  BIN(2) DEF(ODP) POS(159);
DCL DD ODP-RELEASE-NBR  BIN(2) DEF(ODP) POS(161);
DCL DD ODP-OPEN-POSN     CHAR(1) DEF(ODP) POS(163);
DCL DD ODP-OVR-REC-LEN  BIN(2) DEF(ODP) POS(164);

```



```

DCL DD ODP-COM-DEV-CNT    BIN(2)  DEF(ODP) POS(166);

DCL DD ODP.INPUT-BPCA     BIN(4)  DEF(ODP) POS(168);
DCL DD ODP.OUTPUT-BPCA    BIN(4)  DEF(ODP) POS(172);
DCL DD *                   CHAR(1) DEF(ODP) POS(176);

DCL SPCPTR .OPEN-FEEDBACK;
DCL DD OPEN-FEEDBACK CHAR(256) BAS(.OPEN-FEEDBACK);
DCL DD ODP-TYPE          CHAR(2)  DEF(OPEN-FEEDBACK) POS( 1);
DCL DD ODP-FILE          CHAR(10) DEF(OPEN-FEEDBACK) POS( 3);
DCL DD ODP-LIBRARY       CHAR(10) DEF(OPEN-FEEDBACK) POS(13);
DCL DD ODP-SPOOL-FILE    CHAR(10) DEF(OPEN-FEEDBACK) POS(23);
DCL DD ODP-SPOOL-LIB     CHAR(10) DEF(OPEN-FEEDBACK) POS(33);
DCL DD ODP-SPOOL-NBR     BIN(2)   DEF(OPEN-FEEDBACK) POS(43);
DCL DD ODP-MAX-RCD-SIZE  BIN(2)   DEF(OPEN-FEEDBACK) POS(45);
DCL DD ODP-MAX-KEY-SIZE  BIN(2)   DEF(OPEN-FEEDBACK) POS(47);
DCL DD ODP-MEMBER        CHAR(10) DEF(OPEN-FEEDBACK) POS(49);
DCL DD *                  BIN(4)   DEF(OPEN-FEEDBACK) POS(59);
DCL DD *                  BIN(4)   DEF(OPEN-FEEDBACK) POS(63);
DCL DD ODP-DEVICE-TYPE   BIN(4)   DEF(OPEN-FEEDBACK) POS(67);
DCL DD *                  CHAR(3)  DEF(OPEN-FEEDBACK) POS(69);
DCL DD ODP-NBR-LINES     BIN(2)   DEF(OPEN-FEEDBACK) POS(72);
DCL DD ODP-NBR-COLUMNS  BIN(2)   DEF(OPEN-FEEDBACK) POS(74);
DCL DD ODP-NBR-REC-OPEN  BIN(4)   DEF(OPEN-FEEDBACK) POS(76);
DCL DD ODP-ACCESS-TYPE   CHAR(2)  DEF(OPEN-FEEDBACK) POS(80);
DCL DD ODP-DUPL-KEY       CHAR(1)  DEF(OPEN-FEEDBACK) POS(82);
DCL DD ODP-SOURCE-FILE   CHAR(1)  DEF(OPEN-FEEDBACK) POS(83);
DCL DD *                  CHAR(10) DEF(OPEN-FEEDBACK) POS(84);
DCL DD *                  CHAR(10) DEF(OPEN-FEEDBACK) POS(94);
DCL DD ODP-VOL-LABEL     BIN(2)   DEF(OPEN-FEEDBACK) POS(104);
DCL DD ODP-MAX-RCD-BLK   BIN(2)   DEF(OPEN-FEEDBACK) POS(106);
DCL DD ODP-OVERFLOW-LN   BIN(2)   DEF(OPEN-FEEDBACK) POS(108);
DCL DD ODP-BLK-RCD-INCR  BIN(2)   DEF(OPEN-FEEDBACK) POS(110);
DCL DD *                  BIN(4)   DEF(OPEN-FEEDBACK) POS(112);
DCL DD ODP-MISC-FLAGS1   CHAR(1)  DEF(OPEN-FEEDBACK) POS(116);
DCL DD ODP-REQUESTER     CHAR(10) DEF(OPEN-FEEDBACK) POS(117);
DCL DD ODP-OPEN-FILES    BIN(2)   DEF(OPEN-FEEDBACK) POS(127);
DCL DD *                  BIN(2)   DEF(OPEN-FEEDBACK) POS(129);
DCL DD ODP-NBR-FILES     BIN(2)   DEF(OPEN-FEEDBACK) POS(131);
DCL DD ODP-MISC-FLAGS2   CHAR(1)  DEF(OPEN-FEEDBACK) POS(133);
DCL DD ODP-OPEN-ID       CHAR(2)  DEF(OPEN-FEEDBACK) POS(134);
DCL DD ODP-MAX-FMT-SIZE  BIN(2)   DEF(OPEN-FEEDBACK) POS(136);
DCL DD *                  BIN(2)   DEF(OPEN-FEEDBACK) POS(138);
DCL DD ODP-MISC-FLAGS3   CHAR(1)  DEF(OPEN-FEEDBACK) POS(140);
DCL DD *                  CHAR(6)  DEF(OPEN-FEEDBACK) POS(141);
DCL DD ODP-NBR-DEVICES   BIN(2)   DEF(OPEN-FEEDBACK) POS(147);
DCL DD ODP-DEV-LIST...   CHAR(1)  DEF(OPEN-FEEDBACK) POS(149);

DCL SPCPTR .IO-FEEDBACK;
DCL DD IO-FEEDBACK CHAR(512) BAS(.IO-FEEDBACK);
DCL DD IO-FILE-DEPNT-AREA BIN(2)  DEF(IO-FEEDBACK) POS( 1);
DCL DD IO-WRITE-COUNT     BIN(4)  DEF(IO-FEEDBACK) POS( 3);
DCL DD IO-READ-COUNT      BIN(4)  DEF(IO-FEEDBACK) POS( 7);
DCL DD IO-UPDATE-COUNT    BIN(4)  DEF(IO-FEEDBACK) POS(11);
DCL DD IO-OTHER-COUNT     BIN(4)  DEF(IO-FEEDBACK) POS(15);
DCL DD *                  CHAR(1)  DEF(IO-FEEDBACK) POS(19);
DCL DD IO-CUR-OPERATION   CHAR(1)  DEF(IO-FEEDBACK) POS(20);
DCL DD IO-RECORD-FORMAT   CHAR(10) DEF(IO-FEEDBACK) POS(21);
DCL DD IO-DEVICE-CLASS    CHAR(2)  DEF(IO-FEEDBACK) POS(31);
DCL DD IO-DEVICE-NAME     CHAR(10) DEF(IO-FEEDBACK) POS(33);
DCL DD IO-RECORD-LENGTH   BIN(4)  DEF(IO-FEEDBACK) POS(43);
DCL DD *                  CHAR(80) DEF(IO-FEEDBACK) POS(47);
DCL DD IO-RECORDS-IN-BLOCK BIN(2)  DEF(IO-FEEDBACK) POS(127);
DCL DD IO-RCD-FORMAT-LENGTH BIN(2) DEF(IO-FEEDBACK) POS(129);
DCL DD *                  CHAR(2)  DEF(IO-FEEDBACK) POS(131);
DCL DD IO-BLOCK-COUNT     BIN(4)  DEF(IO-FEEDBACK) POS(133);
DCL DD *                  CHAR(8)  DEF(IO-FEEDBACK) POS(137);

/*****/

DCL INSPTR .PROCESS-THIS-ODP;
ENTRY      PROCESS-THIS-ODP INT;
ADDSP      .OPEN-FEEDBACK, .ODP, ODP.OPEN-FEEDBACK;
SKIP-IF-DISPLAY-FILE:
CMPBLA(B)  ODP-TYPE, "DS"/EQ(.PROCESS-THIS-ODP);
SKIP-IF-LOGICAL-FILE:
TSTBUM(B)  ODP-MISC-FLAGS1, X'08'/ONES(.PROCESS-THIS-ODP);

HAVE-DATABASE-OR-SPOOLFILE:
ADDSP      .IO-FEEDBACK, .ODP, ODP.IO-FEEDBACK;
CMPBLA(B)  IO-CUR-OPERATION, X'00'/EQ(.PROCESS-THIS-ODP);

```



CPYBLA	ENTRY-OPERATION,	IO-CUR-OPERATION;
CPYNV	ENTRY-COUNT,	IO-WRITE-COUNT;
ADDN(S)	ENTRY-COUNT,	IO-READ-COUNT;
ADDN(S)	ENTRY-COUNT,	IO-UPDATE-COUNT;
ADDN(S)	ENTRY-COUNT,	IO-OTHER-COUNT;
CMPNV(B)	ENTRY-COUNT,	O/EQ(. PROCESS-THIS-ODP);
CPYBLA	ENTRY-LIBRARY,	ODP-LIBRARY;
CPYBLA	ENTRY-FILE,	ODP-FILE;
CPYBLA	ENTRY-MEMBER,	ODP-MEMBER;
CMPNV(B)	LAST-ENTRY-NBR,	100000/NLO(. PROCESS-THIS-ODP);
ADDN(S)	LAST-ENTRY-NBR,	1;
CPYBWP	ENTRIES(LAST-ENTRY-NBR),	THE-ENTRY;
B	.	PROCESS-THIS-ODP;

## Chapter 22

### How to Generate a Truly Random Number

#### ***What is a Random Number?***

The traditional mathematical definition of randomness is based on Kolmogorov-Chaitin complexity, which defines a random string as one, which has no shorter description than the string itself. Now, is “7” random? Well, can you find a shorter description? Is Archimedes’ constant (our old friend  $\pi$  from Chapter 19) a random number? The description in Chapter 19 is certainly shorter than the infinitude of seemingly random digits of  $\pi$ . Many statistical tests exist for the purpose of seeing if a pattern exists in the purported randomness. “7” may fail some of these tests, while  $\pi$  seems to pass. I guess that part of the answer depends on for what *purpose* you require randomness. We’ll offer the following definition of a random number: it is a number that I can compute but that you cannot. Such a number has surprisingly practical uses.

#### ***Use of Randomness***

Sequences that seem random (and generally in reality are *pseudo* random) are needed for a wide variety of purposes. They are used for unbiased sampling in the Monte Carlo method, and to imitate stochastic natural processes. They are used in implementing randomized algorithms that require arbitrary choices. And their perceived unpredictability is used in games of chance, and in data encryption and secure communications.

To generate a random sequence on a digital computer, one starts with a certain *seed*, then iteratively applies some transformation to it, progressively extracting as long as possible a random sequence. In general, one considers a sequence “random” if no patterns can be recognized in it, no predictions can be made about it, and no simple description of it can be *found* by an adversary. But, purely by nature of the fact that the sequence can be generated by iteration of a definite transformation, then a simple description of it certainly does exist.

Most current practical random sequence generation computer programs are based on linear congruence relations (like the one we used in Chapter 10), or linear feedback shift registers (analogous to linear cellular automata). The linearity and simplicity of these systems lead to efficient algebraic algorithms for predicting the sequences (or deducing their seeds), and limits their degree of randomness unless they are *re-seeded* often. Their usefulness comes down to the generation of good random seeds.

#### ***Bad Seeds***

As the World Wide Web was gaining broad public appeal, the need for secure transmittal of payment information (such as credit-card numbers) became evident. Netscape’s browser began to use the Secure Sockets Layer (SSL) for such transactions. Basically, SSL protects communications by encrypting messages with a secret key - a large, random number. The security of SSL depends crucially on the unpredictability of this number. In 1995, Goldberg and Wagner showed that the algorithm that Netscape used to compute this “unknowable” seed was deeply flawed.

Because Netscape would not release detailed information about how the seed was derived (an important point that we shall return to), Goldberg and Wagner resorted to the (honorable) task of reverse-engineering Netscape’s algorithm by manually decompiling the executable program. Here is what they found:

```
global variable seed;
RNG_CreateContext() {
    (Seconds, microseconds) = time of day' /* time elapsed since 1970 */
    pid = process ID;
    ppid = parent process ID;
    a = mklcpr (microseconds);
    b = mklcpr (pid + seconds + (ppid << 12));
    seed = MD5 (a, b);
}

mklcpr (x) { return ((0xDEECE66D * x + 0x2BBB62DC) >> 1); }
```

The *mk1cpr* function just scrambles the input a bit, and MD5 is the well-known hashing function. The seed generated depends only on the values of *a* and *b*, which in turn depend on just the time of day, the process ID, and the parent process ID. If an adversary can predict these three values, the seed can be computed and the whole scheme is compromised. The time of day can be known to within a certain precision, often better than a second, which means that there are only one million possible choices for the microsecond part (and often a lot less because the clocks on most computer systems do not have true microsecond resolution). If you are running on the same machine that is generating the seed, the process IDs will be known, and, if you are not, it is usually possible to guess their values. Firstly, *ppid* is often 1, or, if not, then it is usually just a bit smaller than the *pid*. The *pids* are normally not considered secret and are often present in message packets from the system. Goldberg and Wagner showed that one could deduce the seed in less than a minute.

Shortly afterwards, it was discovered that Kerberos V4 (another secure protocol) suffered from the same problem, maybe even worse than Netscape since it used the standard Unix *random()* function instead of the much better MD5. At about the same time, it was announced that although the *MIT-MAGIC-COOKIE-1* key had 56 bits, only 256 seed values were possible because of poor use of the *rand()* function:

```
key = rand() % 256; /* take the remainder after dividing by 256 */
```

Similar examples of flawed thinking and sloppy implementation abound, e.g. Sun derived NFS file handles (which serve as tokens to control access to a file and therefore need to be unpredictable) from the traditional *pid* and time-of-day, but forgot to load the time-of-day variable, and on and on.

The conclusion is that generating good random numbers is *hard* and that in spite of that (or perhaps, because of that!) it is often done poorly. As Donald Knuth once quipped: “A random number generator should not be chosen at random”. A good random number process is not just a complicated or (as many would think sufficient) an obscure (“secret”) procedure. Netscape, IBM and many, many others have fallen into this trap that the process must be kept secret. But we must heed Kerckhoffs’ second principle: “compromise of the system should not inconvenience the correspondents”. That is: the system should remain secure even if all the details about its inner workings, down to the actual source code, are known to all, and even if the adversaries have access to the very system producing the randomness.

## **The Entropy Pool**

One solution to the randomness problem is to maintain a pool of information pertaining to physical parameters, properties, and activity of the system. Anything that is determined by external factors can be – and should be – used as input to the pool, such as the time between keystrokes, the timing of disk interrupts, number of network packages arrived, and the like. On a multi-user system, such as the AS/400, the number of page faults, the number of disk read/writes, the time to wait until eligible to get the next time slice, and other such information depends on the overall activity in a manner that is hard to predict or precisely control. These things can go into the pool too.

The word *entropy* is often used as synonymous with randomness and the pool of randomness information is often called the entropy pool. One could have several entropy pools: a fast pool that contains “distilled” values from the other, slower, pools, and a number of ever slower pools containing further data about usage statistics, event occurrences, etc, being continuously collected and percolating up to the faster pools. When you need a random number or a seed for a random number generator, you “consult” the fast pool. Typically a cryptographically sound “message digest”, such as MD5 or SHA (the Secure Hash Algorithm) is computed over the pool and used as your next random number or seed. In this chapter we shall describe the construction of an entropy pool for the AS/400.

## **Sources of Entropy**

We shall utilize the following sources of randomness (or entropy):

- User input (e.g. pass phrases)
- Timing information, such as time of day and processor time used
- Resource Management Data collected by the system
- Variation of *when* the pool collector runs

Some of this information is also available to an adversary, but not at the same time, so the data will be somewhat different, that is: an adversary will have to try a *range* of values instead of knowing exactly what the value is. By collecting a wide variety of data, all of which can only be known to the adversary as a range rather than as definite values, we enlarge the *search space* from which our values can be found. If this process goes on for long enough time, it begins to become unfeasible to guess which values within the compounded sets of ranges we are actually using. This “shotgun” approach tends to make life miserable for a would-be adversary.

## Timing Information

The Materialize Process Attributes instruction, **MATPRATR**, has an option to return the amount of processor time (“CPU-time”) used until now by the current thread. An adversary could be monitoring my invocation stack and ask for the same information at the time the entropy collector runs. Because of the dynamic nature of the situation he is likely to get only an approximation to the result I would get. This is all we need: to force him to consider a range of values. All those ranges multiply together to form a large space to search. Here is how to retrieve the processor time:

```
DCL SPCPTR .P24-ATTR INIT(P24-ATTR);
DCL DD P24-ATTR CHAR(128) BDRY(16);
DCL DD P24-MAX-SIZE BIN(4) DEF(P24-ATTR) POS( 1);
DCL DD P24-ACT-SIZE BIN(4) DEF(P24-ATTR) POS( 5);
DCL DD P24-CPU-TIME-USED CHAR(8) DEF(P24-ATTR) POS(19);

ENTRY GET-THREAD-ACTIVITY INT;
  CPYNV P24-MAX-SIZE, 128;
  MATPRATR .P24-ATTR, *, X'24'; /* THREAD ACTIVITY */

  CPYBLA TIME-VALUE, P24-CPU-TIME-USED;
  CALLI ADD-TIME-TO-ENTROPY, *, .ADD-TIME-TO-ENTROPY;
```

The TIME-VALUE is in the standard 64-bit timestamp format that we explored in Chapter 4:

```
DCL DD TIME-VALUE CHAR(8);
DCL DD TIME-VALUE-HI BIN(4) UNSGND DEF(TIME-VALUE) POS(1);
DCL DD TIME-VALUE-LO BIN(4) UNSGND DEF(TIME-VALUE) POS(5);
```

We convert the timestamp to a number as we did in Chapter 4:

```
DCL DD TIMESTAMP PKD(21,0); /* CAN HOLD UNSIGNED 64-BIT INTEGER */
DCL DD TWO**32 PKD(11,0) INIT(P' 4294967296');
DCL DD TWO**15 PKD(11,0) INIT(P' 32768');

DCL INSPTR .ADD-TIME-TO-ENTROPY;
ENTRY ADD-TIME-TO-ENTROPY INT;
  MULT TIMESTAMP, TIME-VALUE-HI, TWO**32;
  ADDN(S) TIMESTAMP, TIME-VALUE-LO;
  DIV ENTROPY-VALUE, TIMESTAMP, TWO**15;
```

We divide by  $2^{15}$  because the lower 15 bits are all zeroes anyway. The result is what I call the **ENTROPY-VALUE** for that piece of information. We finally divide by  $2^{31}$  and use the remainder (stored as an unsigned 4-byte value) as the **bits** to add to the entropy pool:

```
DCL DD ENTROPY-BITS BIN(4) UNSGND;
DCL DD ENTROPY-VALUE PKD(31,0);

DCL INSPTR .ADD-TO-ENTROPY-POOL;
ENTRY ADD-TO-ENTROPY-POOL INT;
  REM ENTROPY-BITS, ENTROPY-VALUE, TWO**31;
  CPYBLA ENTROPY-POOL(POOL-POSITION: 4), ENTROPY-BITS;
```

## Entropy Pool Format

The entropy pool holds a number (currently eight) of entropy pool *entries*. Each entry consists of seven pairs of 4-byte values. The first value of the pair is a particular measurement and the second value is the real-time duration between measurements.

Here is a (pseudo) declaration of the structure:

```
DCL DD ENTROPY-POOL;
  DCL DD ENTROPY-POOL-ENTRY (8);
    DCL DD ENTROPY-POOL-ENTRY-PAIR (7);
      DCL DD ENTROPY-POOL-ENTRY-PAIR-MEASUREMENT BIN(4) UNSGND;
      DCL DD ENTROPY-POOL-ENTRY-PAIR-DURATION BIN(4) UNSGND;
```

To make it harder for the adversary to replicate the entropy pool, the seven measurements in an entry are not made at the same time. They are staggered in time by a variable amount (the *yield* time) as explained below. Each time the pool is consulted, a new entry is added to the pool with a new set of measurements. When the pool overflows (it only holds eight entries, so that happens quickly), the oldest entry is discarded to make room for the new entry. In this way the pool is constantly renewed. We include an option to flush the pool and start with a clean slate.

## The Yield Time

The **YIELD** MI-instruction allows us to introduce considerable variability in *when* we execute a particular piece of code. It works like this: upon execution of **YIELD**, the dispatching algorithm is run. If another thread of equal or higher priority is eligible to run, then one of these threads is chosen and dispatched to run, and our thread will wait until it's its turn again. Otherwise, the current thread resumes execution.

We could execute **YIELD** several times in a row, hoping that some other thread will run so that it will take some time before we are given control again. We could even compute a hash of the current contents of the entropy pool, form an integer from some of the middle bits of the hash-value, divide that integer by a parameter MAX-YIELDS and use the remainder to determine how many times to execute the **YIELD** instruction. We measure the total amount of real time that this whole process takes (the *yield* time) and use *that* as the duration part of the entropy pair. The yield time will depend on other activities going on at the same time, and is hard to predict on a busy system. In all fairness, we should note that the yield time could be manipulated somewhat by an adversary because he could cause some of that “other activity”.

The code reads:

```
REM          TIMES-TO-YIELD, SHA-HASH-MIDDLE, CUR-MAX-YIELDS;
MATMATR      MACHINE-ATTR, X'0100';
CPYBLA       TIME-BEFORE, MAT-TIMESTAMP; /* clock value before the loop */

: CPYNV      LAST-YIELDS, TIMES-TO-YIELD;
  YIELD;    SUBN(SB) TIMES-TO-YIELD, 1/HI (==1);

MATMATR      MACHINE-ATTR, X'0100';
CPYBLA       TIME-AFTER, MAT-TIMESTAMP; /* clock value after the loop */

SUBLC        TIME-VALUE, TIME-AFTER, TIME-BEFORE;
MULT         TIMESTAMP, TIME-VALUE-HI, TWO**32;
ADDN(S)      TIMESTAMP, TIME-VALUE-LO;
DIV          ENTROPY-VALUE, TIMESTAMP, TWO**15;
```

Finally, we add that value to the pool and return for the next measurement:

```
REM          ENTROPY-BITS, ENTROPY-VALUE, TWO**31;
ADDN(S)      POOL-POSITION, 4;
CPYBLA       ENTROPY-POOL(POOL-POSITION:4), ENTROPY-BITS;
ADDN(SB)     POOL-POSITION, 4/POS(.ADD-TO-ENTROPY-POOL);
```

## Resource Management Data

The Materialize Resource Management Data MI-instruction, **MATRMD**, is used to retrieve activity data collected by the system.

```
MATRMD      .Receiver, Control;
```

The first operand is a space pointer to a receiver area. The second operand is an 8-byte character value. The first byte identifies the generic type of information requested, and the remaining 7 bytes must be binary zeroes. Operand 1 points to the result, which has the following format:

```
DCL DD RESOURCES CHAR(nnnn) BDRY(16);
  DCL DD RMD-MAX-SIZE BIN(4) DEF(RESOURCES) POS( 1) INIT(nnnn);
```

```

DCL DD RMD-ACT-SIZE BIN(4) DEF(RESOURCES) POS( 5);
DCL DD RMD-TIMESTAMP CHAR(8) DEF(RESOURCES) POS( 9);
DCL DD RMD-DATA CHAR(17) DEF(RESOURCES) POS(17);

```

As usual, the receiver contains the number of bytes provided (MAX-SIZE) and actually available for materialization (ACT-SIZE). One could allocate the receiver dynamically depending on the number of bytes available, but we'll keep it simple here and allocate a fixed sized receiver data area (5000 bytes). The TIMESTAMP variable contains the standard 64-bit timestamp for the time at which the data is valid (usually just the current clock value). The detailed format of the data depends on the type of information requested. We'll only document the format of data that we have considered is carrying a reasonable amount of entropy.

## Processor Utilization

Selection option x'01' returns the processor utilization, which is the total amount of processor time used, both by threads and by internal machine functions, since the last IPL:

```

DCL DD RMD-PROCESSOR-TIME CHAR(8) DEF(RMD-DATA) POS( 1);
ENTRY GET-PROCESSOR-UTILIZATION INT;
  CPYBLA WHICH-RESOURCE, X' 01' ; /* PROCESSOR UTIL */
  MATRMD .RESOURCES, WHICH-RESOURCE;

  CPYBLA TIME-VALUE, RMD-PROCESSOR-TIME;
  CALLI ADD-TIME-TO-ENTROPY, *, .ADD-TIME-TO-ENTROPY;
  B .RETURN;

```

## Storage Management Counters

Selection option x'03' returns various counters related to storage management:

- Access Pending is a count of the number of times that a paging request must wait for the completion of a different request for the same page
- Storage Pool Delays is a count of the number of times that threads have been momentarily delayed by the unavailability of a main storage frame in the proper pool
- Directory Look-up operations is a count of the number of times that auxiliary storage directories were interrogated
- Directory Page Faults is a count of the number of times that a page of an auxiliary storage directory was fetched
- Access Group Member Page Faults is a count of the number of times that a page of an object contained in an access group was fetched
- Microcode Page Faults is a count of the number of times that a page of LIC was fetched
- Microtask read operations is a count of the number of reading one or more pages on behalf of a task rather than a thread.
- Microtask write operations is a count of the number of writing one or more pages on behalf of a task rather than a thread.

```

DCL DD RMD-ACCESS-PENDING BIN(2) UNSGND DEF(RMD-DATA) POS( 1);
DCL DD RMD-STG-POOL-DELAYS BIN(2) UNSGND DEF(RMD-DATA) POS( 3);
DCL DD RMD-DIR-LOOKUPS BIN(4) UNSGND DEF(RMD-DATA) POS( 5);
DCL DD RMD-DIR-PFAULTS BIN(4) UNSGND DEF(RMD-DATA) POS( 9);
DCL DD RMD-AG-MBR-PFAULTS BIN(4) UNSGND DEF(RMD-DATA) POS(13);
DCL DD RMD-LIC-PFAULTS BIN(4) UNSGND DEF(RMD-DATA) POS(17);
DCL DD RMD-TASK-READS BIN(4) UNSGND DEF(RMD-DATA) POS(21);
DCL DD RMD-TASK-WRITES BIN(4) UNSGND DEF(RMD-DATA) POS(25);
DCL DD RMD-RESERVED1 BIN(4) UNSGND DEF(RMD-DATA) POS(29);

ENTRY GET-STORAGE-USAGE-COUNTERS INT;
  CPYBLA WHICH-RESOURCE, X' 03' ; /* STORAGE COUNTERS */
  MATRMD .RESOURCES, WHICH-RESOURCE;
  CPYBV ACCUMULATOR, 0;

  ADDN(S) ACCUMULATOR, RMD-RESERVED1;
  ADDN(S) ACCUMULATOR, RMD-ACCESS-PENDING;
  ADDN(S) ACCUMULATOR, RMD-STG-POOL-DELAYS;
  MULT(S) ACCUMULATOR, 10;

```

```

ADDN(S)      ACCUMULATOR, RMD-DIR-PFAULTS;
ADDN(S)      ACCUMULATOR, RMD-AG-MBR-PFAULTS;
ADDN(S)      ACCUMULATOR, RMD-LIC-PFAULTS;
MULT(S)      ACCUMULATOR, 100;

ADDN(S)      ACCUMULATOR, RMD-TASK-READS;
ADDN(S)      ACCUMULATOR, RMD-TASK-WRITES;
MULT(S)      ACCUMULATOR, 1000;
ADDN(S)      ACCUMULATOR, RMD-DIR-LOOKUPS;

CPYNV        ENTROPY-VALUE, ACCUMULATOR;
CALLI        ADD-TO-ENTROPY-POOL, *, .ADD-TO-ENTROPY-POOL;
B            .RETURN;

```

Why didn't we just add up the various counters? Because adding several numbers destroys a lot of the entropy contained in them. If the sum of 5 non-negative numbers is 10, there are 1001 different ways they could sum to 10<sup>1</sup>. All that information is lost by just summing the numbers. If the numbers were correlated, e.g. all had the same high-order digits, e.g. 5000, 5008, 5004, and the like, then the entropy would only derive from the changing low-order digits, so it is only important to not add up the low-order digits. A somewhat crude way of achieving this would be to multiply the running sum by ten before adding in the next number. The above code does something like this, multiplying the ACCUMULATOR by various amounts now and then. What we are trying to achieve is to put 10 pounds in a 5-pound bag. We'll use the same technique for some of the other measurements. If there is a lot of variation in the numbers we are accumulating, this technique is too crude and we lose some entropy, but then we have a lot to lose from, so we come out with a reasonable amount after all. After all this hand waving, it is time to get back to the code.

## Main Storage Pool Information

Selection option x'09' returns various counters related to storage pool management:

- Data Written is the amount of data written from a storage pool to disk to satisfy demand for resources from the pool (i.e. the overhead incurred to service threads)
- Thread Interruptions (Database and Non-DB) is the total number of interruptions to threads.
- Data transferred (Database and Non-DB) is the amount of data transferred from disk to the pool

There are a several pools, and we must iterate through all pools collecting data as we go. Although we lose entropy by adding the numbers for all pools, we retain the entropy associated with the fact that that is activity on the pools:

```

DCL DD RMD-NBR-OF-POOLS      BIN(2)  UNSGND DEF(RMD-DATA) POS( 5);
DCL DD RMD-POOLS(100)      CHAR(32)  DEF(RMD-DATA) POS(17);

DCL DD FOR-THIS-POOL PKD(21,0);
DCL DD POOL-NBR BIN(2) UNSGND;
DCL DD THE-POOL CHAR(32);
DCL DD POOL-DATA-WRITTEN BIN(4) UNSGND DEF(THE-POOL) POS( 5);
DCL DD POOL-THREAD-INT-DB BIN(4) UNSGND DEF(THE-POOL) POS( 9);
DCL DD POOL-THREAD-INT-NON BIN(4) UNSGND DEF(THE-POOL) POS(13);
DCL DD POOL-DATA-POOL-DB BIN(4) UNSGND DEF(THE-POOL) POS(17);
DCL DD POOL-DATA-POOL-NON BIN(4) UNSGND DEF(THE-POOL) POS(21);

ENTRY GET-STORAGE-POOL-ACTIVITY INT;
CPYBLA      WHICH-RESOURCE, X'09';          /* STORAGE POOLS */
MATRMD      .RESOURCES, WHICH-RESOURCE;
CPYNV       ACCUMULATOR, 0;

CPYNV       POOL-NBR, 1;
FOR-NEXT-POOL:
CPYNV       FOR-THIS-POOL, 0;
CMPNV(B)    POOL-NBR, RMD-NBR-OF-POOLS/HI(DONE-WITH-POOL);
CPYBLA      THE-POOL, RMD-POOLS(POOL-NBR);
CMPBLAP(B)  THE-POOL, X'00', X'00' /EQ(=+2);
MULT(RS)    ACCUMULATOR, P'1.2345'; ;

```

<sup>1</sup>  $N$  (non-negative) integers can sum to  $S$  in:  $W = (S + N - 1)! / S! / (N - 1)!$  ways

```

ADDN(S)      FOR-THI S-POOL, POOL-DATA-WRI TTEN;
MULT(S)      FOR-THI S-POOL, 10;

ADDN(S)      FOR-THI S-POOL, POOL-THREAD-I NT-DB;
ADDN(S)      FOR-THI S-POOL, POOL-THREAD-I NT-NON;
MULT(S)      FOR-THI S-POOL, 1000;

ADDN(S)      FOR-THI S-POOL, POOL-DATA-POOL-DB;
ADDN(S)      FOR-THI S-POOL, POOL-DATA-POOL-NON;
ADDN(S)      ACCUMULATOR, FOR-THI S-POOL;
ADDN(SB)     POOL-NBR, 1/POS(FOR-NEXT-POOL);
DONE-WI TH-POOL:

CPYNV        ENTROPY-VALUE, ACCUMULATOR;
CALLI        ADD-TO-ENTROPY-POOL, *, .ADD-TO-ENTROPY-POOL;
B            .RETURN;

```

## Disk Storage Information

Selection option x'12' returns various counters related to disk storage, or ASPs (Auxiliary Storage Pools) as they are called on the AS/400. For each storage unit (i.e. disk) we extract the numbers of the following:

- Blocks transferred to main storage
- Blocks transferred from main storage
- Requests for transfer to main storage
- Requests for transfer from main storage
- Permanent Blocks transferred from main storage
- Requests for Permanent data transfer from main storage
- Times the disk queue was checked to see if it was empty
- Times the disk queue as actually empty

There are generally several disk units, and we must iterate through all units collecting data as we go. Although we lose entropy by adding the numbers for all units, we retain the entropy associated with the fact that that is activity on the units:

```

DCL DD FOR-THI S-DI SK PKD(21,0);
DCL DD DI SK-NBR BI N(2) UNSGND;
DCL DD RMD-NBR-OF-DI SKS BI N(2) UNSGND DEF(RMD-DATA) POS( 3);
DCL DD RMD-OFFSET-TO-DI SKS BI N(4) UNSGND DEF(RMD-DATA) POS(29);

DCL SPCPTR .THE-DI SK;
DCL DD THE-DI SK CHAR(208) BAS(.THE-DI SK);
DCL DD DI SK-UNI T-I D CHAR(22) DEF(THE-DI SK) POS(123);
DCL DD DI SK-BLOCKS-I N BI N(4) UNSGND DEF(THE-DI SK) POS(145);
DCL DD DI SK-BLOCKS-OUT BI N(4) UNSGND DEF(THE-DI SK) POS(149);
DCL DD DI SK-REQI O-I N BI N(4) UNSGND DEF(THE-DI SK) POS(153);
DCL DD DI SK-REQI O-OUT BI N(4) UNSGND DEF(THE-DI SK) POS(157);
DCL DD DI SK-PERM-BLKS-OUT BI N(4) UNSGND DEF(THE-DI SK) POS(161);
DCL DD DI SK-REQI O-PERMS BI N(4) UNSGND DEF(THE-DI SK) POS(165);
DCL DD * CHAR(8) DEF(THE-DI SK) POS(169);
DCL DD DI SK-QUEUE-SAMPLED BI N(4) UNSGND DEF(THE-DI SK) POS(177);
DCL DD DI SK-QUEUE-EMPTY BI N(4) UNSGND DEF(THE-DI SK) POS(181);

ENTRY GET-DI SK-ACTI VI TY-COUNTERS INT;
MULT(R)      ACCUMULATOR, LAST-YI ELDS, PRV-HASH-LO;
CPYBLA       TI ME-VALUE, RMD-TI MESTAMP;
DI V(SB)     TI ME-VALUE-LO, TWO**15/EQ(=+2);
MULT(S)      ACCUMULATOR, TI ME-VALUE-LO;
REM(SB)      LAST-YI ELDS, 4/NZER(DONE-WI TH-DI SKS);

CPYBLA       WHI CH-RESOURCE, X' 12' ; /* DISK UTI LI ZATION */
MATRMD       .RESOURCES, WHI CH-RESOURCE;
CPYNV        ACCUMULATOR, 0;

SETSP        .THE-DI SK, RESOURCES;
ADDSP        .THE-DI SK, .THE-DI SK, RMD-OFFSET-TO-DI SKS;
CPYNV        DI SK-NBR, 1;
FOR-NEXT-DI SK:
CPYNV        FOR-THI S-DI SK, 0;
CMPNV(B)     DI SK-NBR, RMD-NBR-OF-DI SKS/HI (DONE-WI TH-DI SKS);
MULT(RS)     ACCUMULATOR, P' 1.2345' ;

ADDN(S)      FOR-THI S-DI SK, DI SK-QUEUE-SAMPLED;

```



```

ADDN(S)      FOR-THI S-DI SK, DI SK-QUEUE-EMPTY;
MULT(S)      FOR-THI S-DI SK, 100;

ADDN(S)      FOR-THI S-DI SK, DI SK-PERM-BLKS-OUT;
ADDN(S)      FOR-THI S-DI SK, DI SK-REQI O-PERMS;
ADDN(S)      FOR-THI S-DI SK, DI SK-BLOCKS-OUT;
ADDN(S)      FOR-THI S-DI SK, DI SK-REQI O-OUT;
MULT(S)      FOR-THI S-DI SK, 1000;

ADDN(S)      FOR-THI S-DI SK, DI SK-BLOCKS-IN;
ADDN(S)      FOR-THI S-DI SK, DI SK-REQI O-IN;

ADDN(S)      ACCUMULATOR, FOR-THI S-DI SK;
ADDSPP      .THE-DI SK, .THE-DI SK, 208;
ADDN(SB)    DI SK-NBR, 1/POS(FOR-NEXT-DI SK);
DONE-WI TH-DI SKS:

CPYNV      ENTROPY-VALUE, ACCUMULATOR;
CALLI      ADD-TO-ENTROPY-POOL, *, .ADD-TO-ENTROPY-POOL;
B          .RETURN;

```

Because it is fairly expensive in CPU-cycles to get this information, we only refresh the disk information on the average every fourth time, filling in with a timestamp and some function of the previous yield-time the other 75% of the time.

## Activity Level Control Data

Selection option x'16' returns the Activity Level control information. At the MI, activity levels are called "multiprogramming levels" or MPLs. MPLs are grouped into classes. A class determines how many concurrent threads you can have at that level. We aggregate the following information from all classes:

- The number of threads assigned to the class
- The number of active to wait transitions

```

DCL DD FOR-THI S-MPL PKD(21,0);
DCL DD MPL-NBR BIN(2) UNSGND;
DCL DD OFFSET-TO-MPLS BIN(4) INIT(20);
DCL DD RMD-NBR-OF-MPLS BIN(2) UNSGND DEF(RMD-DATA) POS(3);

DCL SPCPTR .THE-MPL;
DCL DD THE-MPL CHAR(32) BAS(.THE-MPL);
DCL DD MPL-CURRENT-MPL BIN(4) UNSGND DEF(THE-MPL) POS(9);
DCL DD MPL-NBR-OF-THREADS BIN(4) UNSGND DEF(THE-MPL) POS(17);
DCL DD MPL-NBR-ACT-WAI T-TRANS BIN(4) UNSGND DEF(THE-MPL) POS(25);

ENTRY GET-MULTI PROGRAMMI NG-LEVELS INT;
CPYBLA      WHI CH-RESOURCE, X'16'; /* MPL CONTROL INFO */
MATRMD      .RESOURCES, WHI CH-RESOURCE;
CPYNV      ACCUMULATOR, 0;

SETSPP      .THE-MPL, RMD-DATA;
ADDSPP      .THE-MPL, .THE-MPL, OFFSET-TO-MPLS;
CPYNV      MPL-NBR, 1;
FOR-NEXT-MPL:
CPYNV      FOR-THI S-MPL, 0;
CMPNV(B)    MPL-NBR, RMD-NBR-OF-MPLS/HI (DONE-WI TH-MPLS);
CMPNV(B)    MPL-CURRENT-MPL, 0/EQ(=+2);
MULT(RS)    ACCUMULATOR, P'1.2345';;

ADDN(S)      FOR-THI S-MPL, MPL-NBR-OF-THREADS;
MULT(S)      FOR-THI S-MPL, 10;
ADDN(S)      FOR-THI S-MPL, MPL-NBR-ACT-WAI T-TRANS;

ADDN(S)      ACCUMULATOR, FOR-THI S-MPL;
ADDSPP      .THE-MPL, .THE-MPL, 32;
ADDN(SB)    MPL-NBR, 1/POS(FOR-NEXT-MPL);
DONE-WI TH-MPLS:

CPYNV      ENTROPY-VALUE, ACCUMULATOR;
CALLI      ADD-TO-ENTROPY-POOL, *, .ADD-TO-ENTROPY-POOL;
B          .RETURN;

```

## Can We Measure the Amount of Entropy in the Pool?

This is really a difficult task. The entropy is a measure of the “unexpectedness” of the data in the pool. Consider the following little anecdote. It is about a ship’s Captain and his First Officer. One day the First Officer learned about some personal problem and got drunk. The Captain wrote in the logbook (that is supposed to record significant events): “Today the First Officer was drunk”. When the First Officer finished his next watch, he wrote in the logbook: “Today the Captain was sober”. The more unexpected the data is, the harder it is for an adversary to guess or predict it.

A crude, but effective, measure of the entropy in the pool is the compressibility of the pool. Completely random data cannot be compressed to a shorter string (The US patent office no longer grants patents on perpetual motion machines, but has recently granted a patent on the mathematically impossible process of compression of truly random data: 5,533,051 "Method for Data Compression". But then, they grant patents on things like “one-click shopping” and US patent 5,443,036 - look it up!). If you compress the pool, the length of the compressed result is usually shorter than the length of the data in the pool, because the pool data is not perfectly random. Now, add another entry to the pool (causing the oldest entry to be discarded). Compress the pool again. If no new information was added the length of the compressed result stays about the same, because the “new” data is already in the compressor’s dictionary. If, on the other hand the new entry is significantly different from the previous entry, the compression is going to be less good and the length of the compressed result increases.

We learned in Chapter 12 how to use the CPRDATA MI-instruction to compress a string of data. Apply that to the entire pool:

```
DCL SPCPTR . COMPRESS-CONTROL INIT(COMPRESS-CONTROL);
DCL DD COMPRESS-CONTROL CHAR(64) BDRY(16);
DCL DD CMPR-SOURCE-LENGTH BIN(4) DEF(COMPRESS-CONTROL) POS( 1);
DCL DD CMPR-BYTES-AVAILABLE BIN(4) DEF(COMPRESS-CONTROL) POS( 5);
DCL DD CMPR-LENGTH BIN(4) DEF(COMPRESS-CONTROL) POS( 9);
DCL DD CMPR-ALGORITHM BIN(2) DEF(COMPRESS-CONTROL) POS(13);
DCL DD CMPR-MBZ CHAR(18) DEF(COMPRESS-CONTROL) POS(15);
DCL SPCPTR . CMPR-SOURCE DEF(COMPRESS-CONTROL) POS(33)
INIT(ENTROPY-POOL);
DCL SPCPTR . CMPR-RESULT DEF(COMPRESS-CONTROL) POS(49)
INIT(ENTROPY-COMPRESSED);
DCL DD FP BIN(2);
DCL DD SIZE BIN(2);
DCL DD ENTROPY-COMPRESSED CHAR(600);

ENTRY ESTIMATE-ENTROPY-ADDED INT;
CPYV CMPR-SOURCE-LENGTH, POOL-POSITION;
CPYV CMPR-BYTES-AVAILABLE, 600;
CPYV CMPR-ALGORITHM, 2; /* LZ1 */
CPYBREP CMPR-MBZ, X'00';
CPRDATA . COMPRESS-CONTROL;

CPYBREP ENTROPY-COMPRESSED, X'00';
SUBN(S) CMPR-LENGTH, 12; /* - FIXED OVERHEAD */
MULT(S) CMPR-LENGTH, 100; /* to get percentage */
DIV(R) CUR-CONFIDENCE, CMPR-LENGTH, CMPR-SOURCE-LENGTH;
```

After subtracting a 12-byte fixed overhead, we computed the resulting length as a percentage of the input length. The result, **CONFIDENCE**, will serve as a “confidence” indicator. Experiments show that if the compressed length is above 80% of the original length, there is a decent amount of entropy in the pool. We can now *monitor* the entropy of the pool. If the **CONFIDENCE** is too low, we can increase the yield-time thus forcing the entropy collector to run for a longer time, increasing the chances of changes in the Resource Management Data as well as giving us more variation of the duration (more entropy right there) of the collections steps.

## Aging the Entropy Pool

Having calculated the “confidence” number one more task awaits us: if the pool is now filled, we must discard the oldest entry, to make room for the next entry:

```
SUBN CMPNV(B) FP, ENTROPY-POOL-SIZE, POOL-STEP;
POOL-POSITION, FP/LO(.RETURN);
```

```

POOL-IS-FILLED:
  SUBN(S)      POOL-POSITION, POOL-STEP;
  SUBN         SIZE, POOL-POSITION, 1;
  ADDN         FP, POOL-STEP, 1;
  CPYBOLAP     ENTROPY-POOL (1:SIZE), ENTROPY-POOL (FP:SIZE), X'00';
  B            .RETURN;

```

Note the **CPYBOLAP** instruction that works like CPYBLAP, but allows **O**verlapping operands.

Having identified the contents and outlined the management of the entropy pool, we can put it all together in the **MI GETETP** program to be discussed next.

## The Get Entropy Program (**MI GETETP**)

Let's first look at the parameters to **MI GETETP**. The first parameter is the SHA hash value of the entropy pool, or more precisely of the entire internal state of **MI GETETP**, which includes the entropy pool. The hash value is 160 bits (20 bytes) long:

```

DCL SPCPTR .PARM1 PARM;
DCL DD PARM1 CHAR(20) BAS(.PARM1);
DCL DD PARM-HASH CHAR(20) DEF(PARM1) POS(1);

```

The second parameter is a *control* parameter. It has three input fields:

- The value of **MAX-YIELDS** that determines the maximum number of times the program will yield its timeslice
- The **RESTART** parameter where a value of "Y" will flush the pool. A value of "Y" is automatically reset to "N" by **MI GETETP**
- A pass **PHRASE** that will be included in calculation of the hash value.

And two output fields:

- The **CONFIDENCE** indicator
- The number of **CALLS** that have been made to the program.

```

DCL SPCPTR .PARM2 PARM;
DCL DD PARM2 CHAR(90) BAS(.PARM2);
DCL DD PARM-MAX-YIELDS BIN(4) DEF(PARM2) POS( 1); /* DFLT = 1000 */
DCL DD PARM-CONFIDENCE BIN(4) DEF(PARM2) POS( 5); /* > 80%: GOOD */
DCL DD PARM-RESTART CHAR(1) DEF(PARM2) POS( 9); /* Y OR N */
DCL DD PARM-PHRASE CHAR(64) DEF(PARM2) POS(10); /* USER SUPPL. */
DCL DD PARM-CALLS PKD(21,0) DEF(PARM2) POS(74); /* NBR OF CALLS*/
DCL DD * CHAR(6) DEF(PARM2) POS(85); /* UNUSED */

```

The optional third parameter exposes the entire internal state of the program. It is used for debugging only and should be removed from the production version:

```

DCL SPCPTR .PARM3 PARM;
DCL DD PARM3 CHAR(600) BAS(.PARM3);
DCL DD PARM-STATE CHAR(640) DEF(PARM3) POS(1);

DCL OL PARMS(.PARM1, .PARM2, .PARM3) PARM EXT MIN(2);

```

## The Internal State

The **CURRENT** field is a copy of the control parameter:

```

DCL SPCPTR .STATE INIT(STATE);
DCL DD STATE CHAR(640) BDRY(16);
DCL DD CURRENT CHAR(90) DEF(STATE) POS( 1);
DCL DD CUR-MAX-YIELDS BIN(4) DEF(CURRENT) POS( 1) INIT(1000);
DCL DD CUR-CONFIDENCE BIN(4) DEF(CURRENT) POS( 5);
DCL DD CUR-RESTART CHAR(1) DEF(CURRENT) POS( 9) INIT("N");
DCL DD CUR-PHRASE CHAR(64) DEF(CURRENT) POS(10);
DCL DD * CHAR(17) DEF(CURRENT) POS(74);

DCL DD NBR-OF-CALLS PKD(21,0) DEF(STATE) POS( 91) INIT(P'0');
DCL DD NBR-OF-PARMS BIN(2) DEF(STATE) POS(103);
DCL DD POOL-POSITION BIN(4) DEF(STATE) POS(105) INIT(1);
DCL DD LAST-YIELDS BIN(4) DEF(STATE) POS(109) INIT(0);

DCL DD ENTROPY-POOL-SIZE BIN(2) DEF(STATE) POS(113) INIT(500);
DCL DD ENTROPY-POOL CHAR(500) DEF(STATE) POS(115);

```

```

DCL DD POOL-STEP          BIN(2) DEF(STATE) POS(615) INIT(0);
DCL DD PREVIOUS-HASH     CHAR(20) DEF(STATE) POS(621);
DCL DD PRV-HASH-LO       BIN(2) UNSGND DEF(PREVIOUS-HASH) POS(19);

```

```
DCL INSPTR .RETURN;
```

The NBR-OF-CALLS variable is increased by 1 every time the program is called. Its current value is returned to the caller, who can then check that no other program (e.g. a rogue program that is part of the application and runs in the same process as the caller) has accessed the entropy pool.

It is trivial to change the program to have its state information belong to the caller:

```

DCL SPCPTR .PARM3 PARM;
DCL DD PARM3 CHAR(640) BAS(. PARM3);

DCL OL PARMS(. PARM1, . PARM2, . PARM3) PARM EXT MIN(3);

DCL DD STATE CHAR(640) DEF(PARM3) POS(1);
DCL DD CURRENT CHAR(90) DEF(STATE) POS( 1);
...

```

## Generation of Entropy

After some initial house-keeping:

```

ENTRY * (PARMS) EXT:
  STPLLEN      NBR-OF-PARMS;
  CPYBLA       CURRENT, PARM2;          /* get control values */
  ADDN(S)      NBR-OF-CALLS, 1;         /* increment call nbr */
  CPYNV        PARM-CALLS, NBR-OF-CALLS; /* return new value */

  CMPBLA(B)    CUR-RESTART, "Y"/NEQ(GET-MACHINE-ACTIVITY);
  CPYBREP      PREVIOUS-HASH, X'00';    /* wipe the pool if */
  CPYBREP      ENTROPY-POOL, X'00';    /* RESTART = 'Y' */
  CPYNV        POOL-STEP, 0;
  CPYNV        POOL-POSITION, 1;

```

We get a new set of measurements:

```

GET-MACHINE-ACTIVITY:
  CALLI      GET-THREAD-ACTIVITY, *, .RETURN;
  CALLI      GET-PROCESSOR-UTILIZATION, *, .RETURN;
  CALLI      GET-STORAGE-USAGE-COUNTERS, *, .RETURN;
  CALLI      GET-STORAGE-POOL-ACTIVITY, *, .RETURN;
  CALLI      GET-DISK-ACTIVITY-COUNTERS, *, .RETURN;
  CALLI      GET-MULTI PROGRAMMING-LEVELS, *, .RETURN;

  CALLI      ESTIMATE-ENTROPY-ADDED, *, .RETURN;

```

With an updated entropy pool we can compute a new hash value, but first we return updated control-variables:

```

  CMPNV(B)    NBR-OF-PARMS, 3/NEQ(=+4); /* this version only exposes the state */
  CMPBLA(B)    CUR-RESTART, "Y"/EQ(=+3); /* if the caller supplies the previous */
  CMPBLA(B)    PARM-HASH, PREVIOUS-HASH/EQ(=+2); /* value of the entropy hash */
  CPYNV        NBR-OF-PARMS, 2;

  CPYBLA      CUR-RESTART, "N";
  CPYBLA      PARM-RESTART, CUR-RESTART;
  CPYNV        PARM-CONFIDENCE, CUR-CONFIDENCE;

MAKE-FINAL-HASH:
  CPYBREP      CIPHER-WORKAREA, X'00';
  CPYNV        CIPHER-LENGTH, 640;      /* hash is computed over */
  CIPHER       . PARM1, CIPHER-OPTS, .STATE; /* the entire internal state */
  CPYBLA      PREVIOUS-HASH, PARM-HASH;

  CMPNV(B)    NBR-OF-PARMS, 3/NEQ(=+2);
  CPYBLA      PARM3, STATE;

  RTX        *;

```

## The Entropy Monitor Program (*MI ETPMON*)

Programs that generate random output are notoriously difficult to debug; how do you know that the output is correct when its main characteristic is that it be unknowable? The only way I know of is peer review (one of the main purposes of this chapter). It is, however, comforting to have some kind of feeling for what the output looks like and how the program behaves. To this end, we wrote a simple testprogram, **MI ETPMON**, using our screen handler and automatic refresh mechanism to show the entropy pool in real time.

A description of the screen is provided in the include file **MI ETPSCR**. A small problem of possible interest is how to describe the four columns of data to show. Here is the description of the row where the columns start:

```
DCL DD * CHAR(10) INIT("L050010030"); DCL DD * CHAR(30) INIT("CPU time by Thread . . . . .");
DCL DD * CHAR(10) INIT("L050310010"); DCL DD S-VALUE-1ST CHAR(10) INIT(" ");
DCL DD * CHAR(10) INIT("L050430010"); DCL DD S-CURCHG-1ST CHAR(10) INIT(" ");
DCL DD * CHAR(10) INIT("L050550010"); DCL DD S-AVGCHG-1ST CHAR(10) INIT(" ");
DCL DD * CHAR(10) INIT("L050670010"); DCL DD S-MAXCHG-1ST CHAR(10) INIT(" ");
```

In the program we define four arrays defined on the four entries named. Since the size of the description block for the row is 120 bytes, the arrays are defined with an “Array Element Offset” (AEO) of 120:

```
DCL DD S-VALUE(14) CHAR(10) DEF(S-VALUE-1ST) POS(1) AEO(120);
DCL DD S-CURCHG(14) CHAR(10) DEF(S-CURCHG-1ST) POS(1) AEO(120);
DCL DD S-AVGCHG(14) CHAR(10) DEF(S-AVGCHG-1ST) POS(1) AEO(120);
DCL DD S-MAXCHG(14) CHAR(10) DEF(S-MAXCHG-1ST) POS(1) AEO(120);
```

The screen looks like this:

Entropy Pool		Moni tor	2001/01/28 23: 23: 39	
	Val ue	Cur. Change	Avg. Change	Max. Change
CPU time by Thread . . . . .	41020835	13967	28257	108852
Yield time . . . . .	619	471	3004	401076
Internal call count . . . . .	1409	1	1	1
Yield time . . . . .	2641	1065	1920	79762
Total Processor time . . . . .	1453283625	81106	119362	678963
Yield time . . . . .	354	4579	1936	33892
System Storage counters . . . . .	1726376466	100000	296463	19207000
Yield time . . . . .	3027	2853	2092	41257
Storage Pool activity . . . . .	1377222174	1884	18564	590529
Yield time . . . . .	936	4132	2016	29825
Disk activity . . . . .	1119798982	948748708	677840823	2133236063
Yield time . . . . .	3422	212	2251	24077
MPL activity . . . . .	166500272	214	1640922	39156265
Yield time . . . . .	3733	2041	2186	49763
Confidence level . . . . .	82	> 80 is good		2000
SHA Value . . . . .	8503E6BA3F58FDDBA4C510DF3A018C90FA57C9B3			
F3=Exit F5=Refresh F10=Restart F11=Keep good F12=Cancel F19=Start Auto				

And shows in the “value” column the current value of the newest entropy pool entry. The other columns show the magnitude of the latest change, the average change, and the maximum change. Press Enter or F5 to refresh the values (i.e. get a new entry in the pool). F10 flushes the pool and starts over. F11 puts the monitor into a mode where it will try to adjust the yield time so that the confidence value stays above (and actually just above) the 80% level. Finally F19 starts an automatic 3-second refresh cycle.

The program is straightforward and will not be shown in detail here (but you can get it and its screen description from the “programs” download area). We’ll just touch upon one little detail. To generate a new screenful of values we call **MI GETETP**, then test the **CONFIDENCE** level. If it is not above 80% for five consecutive cycles, we double the **MAX-YIELDS** parameter. If it is not below 82% for five consecutive cycles, we halve the **MAX-YIELDS** parameter. We are thus striving to keep the **CONFIDENCE** level between 80 and 82% which gives us a high level of randomness, but also keeps control on the length of time it takes to reach that level:

```
GENERATE-SCREEN:
  CPYNNV          START-POS, CUR-POSITION;
```

```

CALLX      .MI GETETP, MI GETETP, *;

TEST-FOR-LOW:
CMPNV(B)   CONFIDENCE, 80/HI (TEST-FOR-HIGH);
SUBN(SB)   MEASURING-INTERVAL, 1/POS (TEST-FOR-HIGH);
MULT(SB)   MAX-YIELDS, 2/NNAN (SET-MEASURING-INTERVAL);

TEST-FOR-HIGH:
CMPNV(B)   CONFIDENCE, 82/LO (SAVE-NEW-VALUES);
SUBN(SB)   MEASURING-INTERVAL, 1/POS (SAVE-NEW-VALUES);
DIV(S)     MAX-YIELDS, 2;
CMPNV(B)   MAX-YIELDS, ORIGINAL-YIELDS/HI (=+2);
CPYNV      MAX-YIELDS, ORIGINAL-YIELDS;

SET-MEASURING-INTERVAL:
CPYNV      MEASURING-INTERVAL, 5;

```

## Conclusion

This Chapter attempts to solve the following problem: Assume that someone has full access to your source code and to the source of any and all of your other code, can you still produce a number that he cannot guess? It is also assumed that he knows when your program is executing (he may be executing it himself), so you cannot simply use the clock. He could guess at every microsecond in a 1-second window around the time of the run. If you used the time, chances are that one of his guesses would be correct. It doesn't matter that he does not know which one.

## TCP/IP Flaw Because of Lack of Randomness

Just to show how important random numbers are we offer the following quote from the trade press:

*by Dennis Fisher, eWEEK, March 12, 2001*

For the second time in as many months, researchers have found a serious flaw in one of the key pieces of the Internet's software backbone. Security vendor *Guardent Inc.* on Monday announced it has identified a potentially huge problem in the inner workings of TCP (Transmission Control Protocol), one half of the TCP/IP standard that enables Internet traffic to flow across heterogeneous networks. In January, researchers identified several holes in the BIND (Berkeley Internet Name Domain) software that runs most of the Internet's name servers.

The latest problem, which is nearly identical to one found in Cisco Systems Inc.'s IOS software two weeks ago and first reported by *eWEEK*, involves the manner in which machines running TCP select the ISN (Initial Sequence Number). The ISN, a random value known only to the two machines at either end of a TCP session, is used to help identify legitimate packets and prevent extraneous data from muddying a transmission.

ISN values are exchanged by the sending and receiving hosts and are supposed to be chosen randomly. Each successive packet then contains a sequence number that is based on the ISN plus the number of bytes transferred to the receiving host. But if the ISN is not chosen at random or if it is increased by a non-random increment in subsequent TCP sessions, an attacker could guess the ISN, thereby enabling him or her to hijack the session's traffic, inject false packets into the stream or even launch a denial of service attack against individual Web servers.

Despite today's advisory, the INS flaw is hardly a new problem. The architects of the early Internet knew that the lack of randomness in the ISN would be a problem as far back as the mid-1980s and warned of the potential consequences. Indeed, *AT&T Corp.* researchers submitted a paper to the Internet Engineering Task Force in 1996 proposing a fix for the problem.

While they acknowledge that it takes a very knowledgeable cracker to exploit the TCP flaw, *Guardent* officials say it's only a matter of time before someone develops a set of tools to do the job and posts them on the Internet. "The hard part was the reduction of this from theory to practice," said Jerry Brady, vice president of research and development at *Guardent*, in Waltham, Mass. "But if someone makes a tool for this available, it wouldn't take a very experienced person to [launch an attack]." *Guardent* officials alerted *CERT* and affected vendors to the problem before making it public -- still, they are likely to take some heat

for publicizing the flaw before a fix is ready. "We're trying to break new ground here," Brady said. "We were intentionally vague about the details of the problem. We want to work with the vendors to fix this."

## Chapter 23

### Automatic Refresh of Display

#### ***The Automatic Refresh Problem***

For programs that monitor an ongoing activity it is often very convenient to be able to update (“refresh”) the information on a display screen automatically, either at a set interval (like every 30 seconds), or even better, at intervals controlled dynamically by end user of the program. When the user presses a function key, the automatic refresh stops and the user can take control. The classical examples of such a program are “Work with Active Jobs” (**WRKACTJOB**) and “Work with System Activity” (**WRKSYSACT**). In a recent trade rag someone asked: “I’m trying to implement an automatic refreshing screen like that available with the WRKSYSACT command. [...] How does the WRKSYSACT command work?” The answer (by Gary Guthrie) began: “IBM could choose from several methods when implementing the automatic refresh exhibited by WRKSYSACT. It’s likely that whatever method IBM chose isn’t available to you in a high-level language”. This is a sorry state of affairs. In this chapter we’ll solve the problem of automatic refresh.

#### ***DDS Specification for the Display File***

A standard problem with the AS/400 is that most commands and APIs do *too much*. For example, both when you open and when you close a display file, the screen is cleared by default, as well as before every output operation. This was presumably meant to make it easier to program for the display. You have to jump through all kinds of hoops to suppress this default behavior. So, we’ll start by reviewing (quoting from IBM’s manuals) the appropriate DDS keywords that we will need to use for doing what we really want to:

#### **ASSUME (Assume) Keyword**

Use this record-level keyword to specify that OS/400 is to assume that this record is already shown on the display when this file is *opened*. Specify the ASSUME keyword for at least one record format within the display file so that the system does not erase the display when the file is opened. If you use the ASSUME keyword, at least one field in the record must be able to be displayed. This keyword cannot be specified with the USRDFN keyword.

#### **CFnn (Command Function) Keyword**

Use this file- or record-level keyword to specify that the function key specified in the keyword (CF01 through CF24) is available for use. It is to be used as a command function (CF) key to transmit changed data as opposed to a command attention (CA) key, which does not transmit changed data. If a CF keyword is not issued for a given function key, that function key is not available to the user. There are similar keywords (ROLLUP/ROLLEDOWN) for the roll up/down keys.

#### **KEEP (Keep) Keyword**

Use this record-level keyword to keep the display from being deleted when the display file is *closed*. The entire display is kept if any of the records on the display have KEEP specified. The default causes the contents of the entire display to be erased when the file is closed.

#### **OVERLAY (Overlay) Keyword**

Use this record-level keyword to specify that the record format you are defining should appear on the display without the entire display being erased first. Normally, the entire display is deleted on each output operation. The display is always erased on the first output operation after the file is opened, except when both ASSUME and OVERLAY are specified.



## USRDFN (User-Defined) Keyword

Use this record-level keyword to specify that the data for this record is in the form of a user-defined data stream. No fields are valid for this record because the data stream formats the display and most other keywords cannot be used.

## INVITE (Invite) Keyword

Use this file- or record-level keyword to *invite* the device for a later read. To send an invite to a specific device, your program sends an output operation to the device with the INVITE keyword in effect. If the record format used has output-capable fields, the data is sent to the device before the device is invited.

INVITE must be used if the display file can have multiple acquired devices and your program does read from invited devices operations. This is because a read from invited devices to a multiple device display file only returns a record from one of the devices that was invited. If you want all the acquired devices to be able to return data, an output operation with INVITE in effect must be sent to each device before the read from invited devices. *Even if there is only one device acquired on the multiple device file, the device must be invited via INVITE before a read-from-invited-devices operation.*

## Invited Devices

In a typical interactive application, a program writes to a display device and then awaits the user's input by blocking program execution at a read operation. This approach is fine when only one display device is communicating with the program, but it is problematic when the program needs to accept input from multiple devices. The program needs some method for accepting input from any of the devices as data arrives from them, rather than waiting for data from one specific device before awaiting the next specific device's data, and so on.

The solution lies in inviting input from a device, and DDS implements this support with the INVITE keyword. The invite operation sends a request for input to a device and then returns control to your program. This lets you request input from multiple devices and continue processing without waiting for input from each device in turn. After your program invites input from multiple devices, it can perform a *read-from-invited-devices* operation any time you want it to process input from any of the invited devices. The read-from-invited-devices operation waits for the time interval specified in the display file's WAITRCD parameter (for the CRTDSPF, OVRDSPF, and CHGDSPF commands). The wait condition ends in one of the following ways:

- ***Data becomes available from an invited device.*** The program receives the display station name, the results of the operation, and any input data. The device is no longer invited, and you must invite input again if you want to receive any from the device.
- ***No display station invited signal.*** This signal indicates that none of the devices associated with the display file are in the invited condition.
- ***Job ended controlled signal.*** This signal indicates that the job in which the program is running is ending with the controlled option on an ENDJOB (End Job), ENDSBS (End Subsystem), ENDSYS (End System), or PWRDWN SYS (Power Down System) command. This signal occurs only once, and devices remain in the invited condition.
- ***No invited devices have data available signal.*** This signal occurs when no data is available from any of the invited devices, the WAITRCD value is \*IMMED, and none of the previous conditions apply. Devices remain in the invited condition.
- ***Time-out on wait for data from invited devices signal.*** This signal occurs when you specify a number of seconds on the WAITRCD parameter, no data becomes available during that time interval, and none of the previous conditions apply. The devices remain in the invited condition.

Upon expiration of the wait for any of these reasons, your program again receives control and can react accordingly.

Although the INVITE keyword was designed to allow you to manage *multiple* devices for the same display file, it can also be used to implement a time-out for a *single* device. The strategy for an RPG-program to do this would be:

- use the INVITE keyword in the DDS for the record format(s) you want to time out
- set the WAITRCD attribute on the display file to the time interval in seconds to cause a time-out condition
- use the MAXDEV keyword on an F-spec for the display file in RPG with the value of 1
- use the WRITE operation code to write the *record* format in RPG (do not use EXFMT)
- use the READ operation code (with an error indicator) to read the *file* in RPG (time-out will occur only if you read the file instead of the record format).

## General-Purpose Time-out Program

We wish to separate the problem of automatic refresh into two sub-problems:

- Generating the data to show and showing it. The data presumably would be changing with time.
- Waiting until the user presses a command key or the device times out, whichever occurs first.

If the device times out, we can re-generate the data screen and show it again. If the user presses a key, we can react to it immediately. It is the *second* sub-problem that is of interest. We'll write a general-purpose program to call to implement the time-out mechanism. The program could be called with a parameter. If the device timed out before any command key was pressed, x '00' shall be returned as the parameter value. If the user pressed a key, the key value (e.g. its AID value) shall be returned. A more fancy version could also accept the time-out interval as a parameter. Such a program would be a really useful general-purpose tool.

## The DDS for the Invited Device

Our time-out program should have its own display file. Since there, generally, will be data showing on the display and we do not wish to erase or interfere with that data, we need to use the ASSUME, KEEP, and OVERLAY keywords to prevent erasure of the screen. We also want to react to any function key, so we'll use CF01 through CF24. Here is the DDS:

A		CF01 CF02 CF03 CF04
A		CF05 CF06 CF07 CF08
A		CF09 CF10 CF11 CF12
A		CF13 CF14 CF15 CF16
A		CF17 CF18 CF19 CF20
A		CF21 CF22 CF23 CF24
A	R DUMMY	ASSUME KEEP
A		1 2 'dummy'
A	R SCREEN	INVI TE OVERLAY

Recall that the ASSUME keyword must be present for at least one record format within the display file and that at least one field in the record must be able to be displayed. We just specify a *dummy* record with a dummy field that we'll never display.

When you create the display file using CRTDSPF, there are a couple of things to note:

### Create Display File (CRTDSPF)

<b>File</b>	> DTI MEOUT	Name
Library	> LSVALGAARD	Name, *CURLIB
Source file	> QDDSSRC	Name, *NONE
Library	> LSVALGAARD	Name, *LIBL, *CURLIB
Source member	> DTI MEOUT	Name, *FILE
Generation severity level	20	0-30
Flagging severity level	0	0-30
Display device	*REQUESTER	Name, *NONE, *REQUESTER
+ for more values		
Text 'description'	*SRCMBRTXT	
Source listing options		*SRC, *NOSRC, *SOURCE...
+ for more values		
<b>Maximum devices</b>	1	1-256
Enhanced display	*YES	*YES, *NO
Restore display	*NO	*NO, *YES

Defer write . . . . .	*YES	*YES, *NO
Character identifier:		
Graphic character set . . . .	*DEV D	Number, *DEV D, *SYSVAL. . .
Code page . . . . .		Number
Decimal format . . . . .	*JOB	*FILE, *JOB
SFLEND text . . . . .	*MSG	*FILE, *MSG
Maximum file wait time . . . .	*IMMED	Seconds, *IMMED, *CLS
<b>Maximum record wait time</b> . . . .	<b>3</b>	Seconds, *NOMAX, *IMMED
Data queue . . . . .	*NONE	Name, *NONE
Library . . . . .		Name, *LIBL, *CURLIB
Share open data path . . . . .	*NO	*NO, *YES
Sort sequence . . . . .	*JOB	Name, *JOB, *LANGIDSHR. . .
Library . . . . .		Name, *LIBL, *CURLIB
Language ID . . . . .	*JOB	Character value, *JOB
<b>Record format level check</b> . . . .	<b>*NO</b>	*YES, *NO
Authority . . . . .	*LIBCRTAUT	Name, *LIBCRTAUT, *ALL. . .
<b>Replace file</b> . . . . .	<b>&gt; *YES</b>	*YES, *NO

The “Maximum record wait time” (**WAITRCD**) parameter should be set to the number of seconds to wait before timing out. I’ve here set it to **3**. You can change this at runtime with a CHGDSPF or OVRDSPF command, but that is not very elegant. We’ll see a bit later in this chapter how to change the time-out value directly with MI.

## RPG Versions of the Time-Out Program

We write the **SCREEN** record format and read the **DTIMEOUT** file. The AID-key is returned in character position 369 of the feedback area. Here is an RPG-ILE version:

```

Fdtimout cfe          workstn
F                                     maxdev(*file)
F                                     infds(feedbk)
D feedbk              ds          369    369
D aidkey              s          1a
D Paid               *Entry      Plist
C                                     Parm
C                                     write
C                                     read      screen
C                                     read      dtimout
C                                     MoveL    aidkey
C                                     eval     *inlr=*on
C                                     Paid

```

Typical of the “voodoo” surrounding these things, if you omit the **maxdev(\*file)** specification, the device does not time out. The **maxdev** parameter from the display file is 1, but even so it has to be there. The **aidkey** value to return is automatically set to x’00’ if the device timed out. We simply move that to the parameter specified on the entry point.

The equivalent RPG/400 version looks like this:

```

FDTIMEOUTCF E          WORKSTN
F                                     KNUM          1
F                                     KINFDS FEEDBK
I FEEDBK              DS
C                                     369 369 AIDKEY
C      *ENTRY      PLIST
C                                     PARM      PAID      1
C                                     WRITESCREEN
C                                     READ      DTIMEOUT
C                                     MOVE      AIDKEY      PAID
C                                     MOVE *ON      *INLR

```

Why bother with the RPG/400 version? Because we can get an MI-listing spool file of the compiled program which will help us with some investigative work:

==> CRTRPGPGM PGM(*program*) GENOPT(\*LIST)

## Call Stack for WRKACTJOB Command

Although we have a program (even two versions!) that works, we still do not have a real understanding of what goes on. The question still stands: how does a command like WRKACTJOB do its automatic refresh?

Start WRKACTJOB in one session and press F19 to start automatic refresh. Then in another session start WRKACTJOB again and look at the call stack for the first session running WRKACTJOB. You'll see something like this:

Lvl	Procedure	Library	Statement	Instruction
	QCMD	QSYS		043B
	QUI CMENU	QSYS		00C1
1	QUI MNDRV	QSYS		0502
2	QUI MGFLW	QSYS		04B5
3	QUI CMD	QSYS		0419
	QWCCDSAC	QSYS		051A
4	QUI MGFLW	QSYS		03A5
	QUI EXFMT	QSYS		008C
	QUI NMGR	QSYS		021B
	<b>QDMACCI N</b>	<b>QSYS</b>		<b>013A</b>

While WRKACTJOB is waiting for the device to time out or for you to press a command key, it is executing the program **QDMACCI N**. If you check in Appendix D, you'll find that **QDMACCI N** is SEPT entry number 121. One might guess that the program name is derived from "Data Management: ACCEpt INput".

## The **QDMACCI N** Program

We'll not (yet) perform a detailed analysis of the **QDMACCI N** program, but just to whet your appetite a bit, here is the neighborhood of the instruction where the program waits:

```

387E0338 ADDI 3,30,824          0137 ;
440001A1 SCV 13                ; MATMDATA (get clock timestamp)

E8DE0338 LD 6,0X338(30)        0138 ; copy clock value
F8DE0358 STD 6,0X358(30)       ; for use below

E8608130 LD 3,0X8130(0)        0139 ;
38800100 ADDI 4,0,256          ;
440003C1 SCV 30                ; SLIC module #EMMPEVM

E8610160 LD 3,0X160(1)         013A ;
389E0090 ADDI 4,30,144         ;
38BE0350 ADDI 5,30,848         ; use clock value
38DE0590 ADDI 6,30,1424        ;
39400028 ADDI 10,0,40          ; Function 40 = Wait on Event
44000141 SCV 10                ;

E8608130 LD 3,0X8130(0)        013B ;
38800000 ADDI 4,0,0            ;
440003C1 SCV 30                ; SLIC module #EMMPEVM

```

Armed with the fact that **QDMACCI N** is entry 121 in the SEPT we can quickly check if the RPG/400 version of our time-out program uses the same method as WRKACTJOB. Scanning the generated MI-program listing for the string "121" reveals the following occurrence:

```

DCL DD .NULLCL CHAR(1) INIT(X'FF') /*NO CONTROL LIST*/;
...
SETSPP .C001001, .NULLCL /*NO CONTROL LIST*/;
...
..M01002: /*GENERAL GET*/ ;
DCL CON ..ACC BIN(2) INIT(121) /*ETP ACCINPUT*/;
DCL OL ..L01001(.F01UFCB, .C001001);
CPYBWP .I OEPTR, .DMCENTR(..ACC) /*ACCI NPUT*/;
CALLX .I OEPTR, ..L01001, * /*I/O REQUEST*/ ;
CPYBLA .F B10FB, .F BFB10 /*SET FEEDBACK AREA*/;

```

What goes on here is that the RPG program *also* uses QDMACCI N. It is also clear that QDMACCI N is called with two parameters, a User File control Block (UFCB) and an empty control list (refer back to Chapter 12 for more on UFCBs and Control Lists).

Knowing this, we can construct an MI-program duplicating our RPG time-out programs. The body of the code should do as follows (assuming suitable data declarations as used in Chapter 12):

```

OPEN-DI SPLAY:
CPYBWP .ODP, *;
CPYBWP .NEXT-UFCB, *;
CALLX .SEPT(12), OPEN, *;

```

```

WRITE-DISPLAY:
  ADDSPP      .DMDEV, .ODP, DEV-OFFSET;
  CALLX       .SEPT(PUT-FCT), PUT, *;

WAIT:
  CALLX       .SEPT(121), ACC, *; /* ODMACCN */
  ADDSPP      .DEV-DEPEND-FEEDBACK, .IO-FEEDBACK,
  OFFSET-TO-DEV-DEPEND;
  CPYBLA      PARM-AID, DFB-AID;

CLOSE-DISPLAY:
  CALLX       .SEPT(11), CLOSE, *;

```

## Changing the Time-out Value Dynamically

It seems clear that the time-out value must be stored somewhere in the display file object. But where? Let's compile the display file with WAITRCD(4660). The decimal value 4660 is nothing but x'1234', giving us an eye-catcher when we look at the object. Here is a snapshot of the content of the display file (type/subtype x'1901'):

Address	342703C101	000000			
000000	00010008	00808000	342703C1	01000000	.....00....A....
000010	C0010000	00000000	342703C1	01000100	{.....A....
000020	80001901	C4E3C9D4	C5D6E4E3	40404040	0...DTIMEOUT
000030	40404040	40404040	40404040	40404040	
000040	40408000	00000F00	00000008	3F100301	0.....
...					
000400	00000000	00000064	00000011	12345248	.....A.....ec ← time-out value
000410	0001FFFF	00020001	00015C03	C9C2C3D9	.....*LIBCR
000420	E3C1E4E3	00000000	00010000	00000000	TAUT.....

The time-out value is stored at absolute offset x'40C', which is offset x'30C' within the associated space (at POS(781)). We can now resolve a system pointer to the display file object, set a space pointer to its associated space, and define DSPF-DELAY as BIN(2) at POS(781) based on the space pointer to the associated space:

```

DCL SYSPTR .DSPF-SYSTEM;
DCL SPCPTR .DSPF-SPACE;

DCL DD DSPF-SPACE CHAR(1024) BAS(.DSPF-SPACE);
DCL DD DSPF-DELAY BIN(2) DEF(DSPF-SPACE) POS(781);

SET-DELAY:
  CPYBLA      RESOLVE-TYPE, X'1901';
  CPYBLAP     RESOLVE-NAME, FILE-NAME, " ";
  RSLVSP      .DSPF-SYSTEM, RESOLVE, *, *;
  SETSPFPF    .DSPF-SPACE, .DSPF-SYSTEM;
  CPYNV       DSPF-DELAY, PARM-DELAY; /* change to value in parameter */

```

For this to work our program must have the *system state* attribute. If you don't need this capability, just remove the SET-DELAY code.

## The MI / NVDSP Invited Display Program

With the above code fragments and suitable data declarations we can now complete our display time-out program. Note that the delay value (time-out value) is passed to the program and that the AID key is returned:

```

DCL SPCPTR .PARM1 PARM;
DCL DD PARM1 CHAR(3) BAS(.PARM1);
DCL DD PARM-DELAY BIN(2) DEF(PARM1) POS(1); /* time-out value in seconds */
DCL DD PARM-AID CHAR(1) DEF(PARM1) POS(3); /* returns x'00' if timed out */
DCL OL PARMS(.PARM1) PARM EXT MIN(1);

DCL SPCPTR @SEPT BASPCO;
DCL SPCPTR .SEPT(6440) BAS(@SEPT);

DCL SPCPTR .UFCB INIT(UFCB);
DCL DD UFCB CHAR(210) BDRY(16);
DCL SPCPTR .ODP DEF(UFCB) POS( 1);
DCL SPCPTR .INBUF DEF(UFCB) POS(17);
DCL SPCPTR .OUTBUF DEF(UFCB) POS(33);

```

```

DCL SPCPTR .IO-FEEDBACK DEF(UFCB) POS( 65);
DCL SPCPTR .NEXT-UFCB DEF(UFCB) POS( 81);

DCL DD FILE-NAME CHAR(10) DEF(UFCB) POS(129) INIT("DTIMEOUT");
DCL DD LIB-ID BIN ( 2) DEF(UFCB) POS(139) INIT(-75);
DCL DD LIBRARY CHAR(10) DEF(UFCB) POS(141) INIT("*LIBL");
DCL DD MBR-ID BIN ( 2) DEF(UFCB) POS(151) INIT(-71);
DCL DD MEMBER CHAR(10) DEF(UFCB) POS(153) INIT("*FIRST");

DCL DD FLAGS-1 CHAR( 1) DEF(UFCB) POS(175) INIT(X' 80');
DCL DD FLAGS-2 CHAR( 1) DEF(UFCB) POS(176) INIT(X' 00');

DCL DD NO-MORE-PARAMS BIN ( 2) DEF(UFCB) POS(209) INIT(32767);

DCL SPC IO-FEEDBACK BAS(.IO-FEEDBACK);
DCL DD OFFSET-TO-DEV-DEPEND BIN (2) DIR;

DCL SPCPTR .DEV-DEPEND-FEEDBACK;
DCL SPC DEV-DEPEND-FEEDBACK BAS(.DEV-DEPEND-FEEDBACK);
DCL DD DFB-FLAGS CHAR(2) DEF(DEV-DEPEND-FEEDBACK) POS(1);
DCL DD DFB-AID CHAR(1) DEF(DEV-DEPEND-FEEDBACK) POS(3);

DCL SPC ODP BAS(.ODP);
DCL DD * CHAR(16) DIR;
DCL DD DEV-OFFSET BIN ( 4) DIR;

DCL SPCPTR .DMDEV;
DCL SPC FUNCTION-LIST BAS(.DMDEV);
DCL DD MAX-DEVICE BIN ( 2) DIR;
DCL DD NBR-DEVICES BIN ( 2) DIR;
DCL DD DEVICE-NAME CHAR(10) DIR;
DCL DD WORKAREA-OFFSET BIN ( 4) DIR;
DCL DD WORKAREA-LENGTH BIN ( 4) DIR;
DCL DD LUD-PTR-INDEX BIN ( 2) DIR;

DCL DD GET-FCT BIN ( 2) DIR;
DCL DD GET-DIR BIN ( 2) DIR;
DCL DD GET-KEY BIN ( 2) DIR;
DCL DD * BIN ( 2) DIR;
DCL DD PUT-FCT BIN ( 2) DIR;
DCL DD PUT-GET BIN ( 2) DIR;

DCL SPCPTR .PUTOPT INIT(PUTOPT);
DCL DD PUTOPT CHAR(4);
DCL DD PUT-OPTION-BYTE CHAR(1) DEF(PUTOPT) POS(1) INIT(X' 00');
DCL DD PUT-SHARE-BYTE CHAR(1) DEF(PUTOPT) POS(2) INIT(X' 00');
DCL DD PUT-DATA-BYTE CHAR(1) DEF(PUTOPT) POS(3) INIT(X' 00');
DCL DD PUT-DEVICE-BYTE CHAR(1) DEF(PUTOPT) POS(4) INIT(X' 05');

DCL SPCPTR .FORMAT INIT(FORMAT);
DCL DD FORMAT CHAR(14);
DCL DD FORMAT-ID CHAR( 1) DEF(FORMAT) POS( 1) INIT(X' 01');
DCL DD FORMAT-SIZE BIN ( 2) DEF(FORMAT) POS( 2) INIT( 10);
DCL DD FORMAT-NAME CHAR(10) DEF(FORMAT) POS( 4) INIT("SCREEN");
DCL DD FORMAT-END CHAR( 1) DEF(FORMAT) POS(14) INIT(X' FF');

DCL SPCPTR .NOFORMAT INIT(NOFORMAT);
DCL DD NOFORMAT CHAR(1);
DCL DD NOFORMAT-END CHAR(1) DEF(NOFORMAT) POS(1) INIT(X' FF');

DCL OL OPEN (.UFCB);
DCL OL PUT (.UFCB, .PUTOPT, .FORMAT);
DCL OL ACC (.UFCB, .NOFORMAT);
DCL OL CLOSE(.UFCB);

DCL SYSPTR .DSPF-SYSTEM;
DCL SPCPTR .DSPF-SPACE;

DCL DD DSPF-SPACE CHAR(1024) BAS(.DSPF-SPACE);
DCL DD DSPF-DELAY BIN(2) DEF(DSPF-SPACE) POS(781);

DCL DD RESOLVE CHAR(34);
DCL DD RESOLVE-TYPE CHAR( 2) DEF(RESOLVE) POS( 1);
DCL DD RESOLVE-NAME CHAR(30) DEF(RESOLVE) POS( 3);
DCL DD RESOLVE-AUTH CHAR( 2) DEF(RESOLVE) POS(33) INIT(X' 0000');

/*****/

ENTRY * (PARMS) EXT;
SET-DELAY:
    CPYBLA RESOLVE-TYPE, X' 1901';

```

```

CPYBLAP      RESOLVE-NAME, FILE-NAME, " ";
RSLVSP       .DSPF-SYSTEM, RESOLVE, *, *;
SETSPFPF     .DSPF-SPACE, .DSPF-SYSTEM;
CPYNV        DSPF-DELAY, PARM-DELAY;

OPEN-DISPLAY:
CPYBWP       .ODP, *;
CPYBWP       .NEXT-UFCB, *;
CALLX        .SEPT(12), OPEN, *;

WRITE-DISPLAY:
ADDSPP       .DMDEV, .ODP, DEV-OFFSET;
CALLX        .SEPT(PUT-FCT), PUT, *;

WAIT:
CALLX        .SEPT(121), ACC, *; /* ODMACCIN */
ADDSPP       .DEV-DEPEND-FEEDBACK, .IO-FEEDBACK,
              OFFSET-TO-DEV-DEPEND;
CPYBLA       PARM-AID, DFB-AID;

CLOSE-DISPLAY:
CALLX        .SEPT(11), CLOSE, *;
RTX          *;

```

## Automatic Refresh Driver Program *MI AUTREF*

We'll use our simple screen handler, **MI SCRNI O** (Chapter 17), to show the screen for the automatic refresh driver program. The screen shows the current date and time, the corresponding timestamp value, and if refresh is active. An input field at the far right allows the time-out value (refresh interval) to be specified. Press F10 to start automatic refresh and any command key to stop automatic refresh. Press Enter for manual refresh:

Testscreen				
2001/01/04	00:31:28	81CFF73FBEEC0000	Y	__3
Press a function key to stop refresh				
F3=Exit F10=Auto refresh F12=Cancel				

The main loop puts the screen, calls the time-out monitor, checks if the AID key returned was x'00', and, if so, continues the loop, otherwise the screen is read to see what to do next. Note how the **S-AUTO-REFRESH** variable keeps track of the state:

```

DI ALG-LOOP:
CALLI        PUT-THE-SCREEN, *, .PUT-THE-SCREEN;
CMPBLA(B)    S-AUTO-REFRESH, "Y"/NEQ(GET-THE-SCREEN);

WAIT-FOR-INVI TED:
CALLX        MI INVDSP, MI INVDSP, *;
CMPBLA(B)    INVI TED-PFKEY, X'00'/EQ(DI ALG-LOOP);
CPYBLA       S-AUTO-REFRESH, "N";
CALLI        PUT-THE-SCREEN, *, .PUT-THE-SCREEN;

GET-THE-SCREEN:
CPYBLA       CTRL-OPCODE, "READ";
CALLX        MI SCRNI O, MI SCRNI O, *;

CHECK-PASSI VE-KEYS:
CMPNV(B)     CTRL-CMD-KEY, 03/EQ(EXIT);
CMPNV(B)     CTRL-CMD-KEY, 10/EQ(AUTOMATI C);
CMPNV(B)     CTRL-CMD-KEY, 12/EQ(CANCEL);

CHECK-ACTI VE-KEYS:
B            DI ALG-LOOP;

AUTOMATI C:
CPYBLA       S-AUTO-REFRESH, "Y";
CVTEFN       INVI TED-DELAY, S-DELAY, *;
B            DI ALG-LOOP;

```

The internal routine **PUT-THE-SCREEN** gets the clock and updates the screen display. The complete program is now straightforward:

```

DCL DD RESOLVE CHAR(34);
DCL DD RESOLVE-TYPE CHAR( 2) DEF(RESOLVE) POS( 1);
DCL DD RESOLVE-NAME CHAR(30) DEF(RESOLVE) POS( 3);
DCL DD RESOLVE-AUTH CHAR( 2) DEF(RESOLVE) POS(33) INIT(X' 0000' );

DCL SPCPTR .CONTROL INIT(CONTROL);
DCL DD CONTROL CHAR(8);
DCL DD CTRL-OPCODE          CHAR(1) DEF(CONTROL) POS(1);
DCL DD CTRL-CMD-KEY          ZND(2,0) DEF(CONTROL) POS(2);
DCL DD CTRL-CURSOR-POSITION CHAR(5) DEF(CONTROL) POS(4);
DCL DD CTRL-CURSOR-ROW ZND(2,0) DEF(CTRL-CURSOR-POSITION) POS(1);
DCL DD CTRL-CURSOR-COL ZND(3,0) DEF(CTRL-CURSOR-POSITION) POS(3);

DCL SPCPTR .SCREEN INIT(SCREEN);
DCL DD SCREEN CHAR(10) INIT("T010310010");
DCL DD * CHAR(10) INIT("Testscreen");
DCL DD * CHAR(10) INIT("L010420000");
DCL DD * CHAR(10) INIT("L030010028");
DCL DD S-DATE-TIME CHAR(28) INIT(" ");
DCL DD * CHAR(10) INIT("L030400016");
DCL DD S-TIMESTAMP CHAR(16) INIT(" ");
DCL DD * CHAR(10) INIT("L030600001");
DCL DD S-AUTO-REFRESH CHAR(1) INIT("N");
DCL DD * CHAR(10) INIT("I030700003");
DCL DD S-DELAY CHAR(3) INIT(" 3");
DCL DD * CHAR(10) INIT("0060081000");
DCL DD S-TEXT CHAR(1000) INIT(" "); /* to ensure large data portion */

DCL DD * CHAR(10) INIT("U230010079");
DCL DD PF CHAR(79) INIT("F3=Exit F10=Auto refresh F12=Cancel");

DCL DD * CHAR(10) INIT(".000000000"); /* END OF SCREEN */

DCL OL MI SCRNI O(.CONTROL, .SCREEN);
DCL SYSPTR .MI SCRNI O;

DCL SPCPTR .INVPARM INIT(INVPARM);
DCL DD INVPARM CHAR(3);
DCL DD INVITED-DELAY BIN(2) DEF(INVPARM) POS(1);
DCL DD INVITED-PFKEY CHAR(1) DEF(INVPARM) POS(3);

DCL OL MI INV DSP(.INVPARM) ARG;
DCL SYSPTR .MI INV DSP;

/*****

ENTRY * EXT;
  CPYBLA      RESOLVE-TYPE, X' 0201' ;
  CPYBLAP     RESOLVE-NAME, "MI SCRNI O", " ";
  RSLVSP      .MI SCRNI O, RESOLVE, *, *;

  CPYBLA      CTRL-OPCODE, "OPEN";
  CALLX       .MI SCRNI O, MI SCRNI O, *;
  CPYBLA      CTRL-CURSOR-POSITION, "01001";

  CPYBLA      S-AUTO-REFRESH, "N";
  CPYBLAP     RESOLVE-NAME, "MI INV DSP", " ";
  RSLVSP      .MI INV DSP, RESOLVE, *, *;

DIALOG-LOOP:
  CALLI       PUT-THE-SCREEN, *, .PUT-THE-SCREEN;
  CMPBLA(B)   S-AUTO-REFRESH, "Y"/NEQ(GET-THE-SCREEN);

WAIT-FOR-INVITED:
  CALLX       .MI INV DSP, MI INV DSP, *;
  CMPBLA(B)   INVITED-PFKEY, X' 00' /EQ(DIALOG-LOOP);
  CPYBLA      S-AUTO-REFRESH, "N";
  CALLI       PUT-THE-SCREEN, *, .PUT-THE-SCREEN;

GET-THE-SCREEN:
  CPYBLA      CTRL-OPCODE, "READ";
  CALLX       .MI SCRNI O, MI SCRNI O, *;

CHECK-PASSIVE-KEYS:
  CMPNV(B)    CTRL-CMD-KEY, 03/EQ(EXIT);
  CMPNV(B)    CTRL-CMD-KEY, 10/EQ(AUTOMATIC);

```



```

        CMPNV(B)      CTRL-CMD-KEY, 12/EQ(CANCEL);

CHECK-ACTIVE-KEYS:
    B                DIALOG-LOOP;

AUTOMATIC:
    CPYBLA           S-AUTO-REFRESH, "Y";
    CVTEFN           INVITED-DELAY, S-DELAY, *; /* see below */
    B                DIALOG-LOOP;

EXIT:
CANCEL:
    CPYBLA           CTRL-OPCODE, "CLOSE";
    CALLX            MI SCRNI O, MI SCRNI O, *;
    RTX              *;

/*****

DCL INSPTR .PUT-THE-SCREEN;
ENTRY      PUT-THE-SCREEN INT;
    CALLI    GET-DATE-TIME, *, .GET-DATE-TIME;
    CPYBRAP  S-DATE-TIME, YMDHMS, " ";
    CVTHC    S-TIMESTAMP, MCH-TIMESTAMP;

    CMPBLA(B) S-AUTO-REFRESH, "Y"/NEQ(=+2);
    CPYBLA    S-TEXT, "Press a function key to stop refresh";
    CMPBLA(B) S-AUTO-REFRESH, "N"/NEQ(=+2);
    CPYBREP   S-TEXT, " ";

    CPYBLA    CTRL-CURSOR-POSITION, "01001";
    CPYBLA    CTRL-OPCODE, "WRITE";
    CALLX     MI SCRNI O, MI SCRNI O, *;
    B         PUT-THE-SCREEN;

DCL SPCPTR .MCH-ATTR INIT(MCH-ATTR);
DCL DD     MCH-ATTR CHAR(16);
    DCL DD  MCH-BYTES-PROVIDED BIN(4) DEF(MCH-ATTR) POS(1) INIT(16);
    DCL DD  MCH-TIMESTAMP     CHAR(8) DEF(MCH-ATTR) POS(9);
    DCL DD  MCH-TIME-HI BIN(4) UNSGND DEF(MCH-TIMESTAMP) POS(1);
    DCL DD  MCH-TIME-LO BIN(4) UNSGND DEF(MCH-TIMESTAMP) POS(5);

DCL INSPTR .GET-DATE-TIME;
ENTRY      GET-DATE-TIME INT;
    [... here follows our trusted timestamp to date conversion routine from Chapter 4 ...]
    B              GET-DATE-TIME;

```

## The CVTEFN MI-Instruction

To convert the time-out interval entered on the screen to a valid number (rather than just a text) we used the “Convert from External Form to Numeric” (**CVTEFN**) instruction:

```
CVTEFN      INVITED-DELAY, S-DELAY, *;
```

This instruction converts the character string given as operand 2 to a numeric value in operand 1. The third operand (if not null) controls stuff like decimal point character and the like. Unfortunately, the conversion is not very robust, *e.g.* trailing spaces are not allowed, but for this test program we live with that.

## The ROLLUP/ROLLDOWN Problem

We may also wish to have the “Roll” keys to be active. We can include them in the DDS at the record level:

A		CF01 CF02 CF03 CF04
A		CF05 CF06 CF07 CF08
A		CF09 CF10 CF11 CF12
A		CF13 CF14 CF15 CF16
A		CF17 CF18 CF19 CF20
A		CF21 CF22 CF23 CF24
A		<b>ROLLUP ROLLDOWN</b>
A	R DUMMY	ASSUME KEEP
A		1 2 'dummy'
A	R SCREEN	INVITE OVERLAY
A		<b>24 80 ' '</b>

←

Unfortunately, they are not allowed *unless* there is at least one field in the record format. We shall solve this by placing a dummy 1-character field at the end of the last line of the screen. This is often where the message line is, so we only sacrifice a little bit of the message line to get ROLLUP/ROLLDOWN to work.

### ***Acknowledgements***

Several people have helped with research for this chapter: Craig Rutledge, Jr., Carsten Flensburg, Buck Calabro, and others. Thanks to all.

Blank

## Chapter 24

### Editing of Numeric Variables

#### *The Importance of Editing*

It has been said that 90% of all computer power is spent on sorting (either directly or indirectly through maintenance of keyed data), and that 90% of the remaining power is spent on editing of numeric data for presentation. One rationale for the use of “zoned” and “packed” data formats was mainly to cut down on the computational cost of editing (dividing by 10 several times for each number is expensive - some early computers did not even have a `DIVIDE` instruction). Another rationale was simply that many early computers mainly worked in decimal anyway (even addresses were decimal).

#### *Edit Codes and Edit Words*

Traditionally, editing numeric values on the AS/400 was specified through the use of *Edit Codes* and *Edit Words*.

“An edit code is a standard description of how a number should be formatted. There are many standard edit codes defined by the system. Users can define several edit codes the way they want with the use of the Create Edit Description (`CRTEDTD`) command. An edit word is a user-defined description of how a number should be formatted. An edit word is usually used when one of the standard edit codes or user-defined edit codes is not sufficient for a particular situation”.

The above quote from one of the RPG manuals already hints that editing using edit codes and edit words is less than satisfactory. In fact, it often seems that these techniques are part of the problem rather than part of the solution. Luckily, the machine has a very powerful **EDIT** MI-instruction that makes the use of the non-intuitive edit codes and edit words superfluous.

#### *COBOL Pictures*

In contrast to RPG, the COBOL language has always had a very simple, visual, and intuitive way of working with editing specifications: the COBOL *Picture*. With only a slight simplification one can state that the **EDIT** instruction’s purpose is to support the COBOL *edited picture* specification. A numeric variable in COBOL can be either in internal numeric or in *edited numeric* format. Let’s look at a few examples. The `IN-DATA` picture is internal numeric, while all the `OUT-DATA` pictures are edited numeric:

```
02  IN-DATA          PIC  S9(7)V99.
02  OUT-DATA1        PIC  -9,999,999.99.
02  OUT-DATA2        PIC  --,---,--9.99.
02  OUT-DATA3        PIC  9,999,999.99-.
02  OUT-DATA4        PIC  Z,ZZZ,ZZ9.99-.

MOVE -1234567.89 TO IN-DATA
MOVE IN-DATA TO OUT-DATA1, OUT-DATA2, OUT-DATA3, OUT-DATA4
DISPLAY      OUT-DATA1, OUT-DATA2, OUT-DATA3, OUT-DATA4      results in:

-1,234,567.89 -1,234,567.89 1,234,567.89- 1,234,567.89-

MOVE -1.23 TO IN-DATA
MOVE IN-DATA TO OUT-DATA1, OUT-DATA2, OUT-DATA3, OUT-DATA4
DISPLAY      OUT-DATA1, OUT-DATA2, OUT-DATA3, OUT-DATA4      results in:

-0,000,001.23 -1.23 0,000,001.23- 1.23-
```

Very little explanation is really needed. The pictures speak for themselves. The only subtlety is that two of the edited pictures (with all the 9s) specify a **fixed** format, while the other two specify a **floating** format. In the floating format (nothing to do with *floating-point* format), the leading blanks and the sign “float” to the right until they meet the first significant digit. Note also the thousand separator commas that are inserted at either fixed places (for fixed formats) or as needed among significant digits (for floating formats). In the following section we’ll see how to instruct the **EDIT** instruction to do its magic.

## The *EDIT* Instruction

The general format of the *EDIT* instruction is:

*EDIT* *T* *Character-Receiver*, *Numeric-Source*, *Edit-Mask*

The *Edit-Mask* controls the editing of the *Numeric-Source* into the *Character-Receiver*. The first step of the editing process is to automatically translate the source number to a packed decimal format large enough to hold the numeric value.

### The Edit Mask

The COBOL move-statement: `MOVE IN-DATA TO OUT-DATA1` generates the following code:

```
DCL CON PICTURE1 CHAR(23) INIT(X' AF40AE60AEAA B36BAEAAAAAA B36BAEAAAAAA B34BAEAAAA' ;  
EDIT OUT-DATA1, IN-DATA, PICTURE1;
```

Although the edit mask (PICTURE1) looks somewhat forbidding it is really just a simple encoding of the COBOL picture:

Edit Mask Part	Explanation for -9, 999, 999. 99
AF 40 AE 60 AE	Positive: <b>space</b> ; Negative: <b>minus sign</b>
AA	9 digit position (significant or not)
B3 6B AE	, separator comma
AA AA AA	999 digit positions
B3 6B AE	, separator comma
AA AA AA	999 digit positions
B3 4B AE	. decimal point
AA AA	99 decimals

The edit mask contains both *control* characters and *data* characters. The character x'AE' is used as an "End-of-String" character (EOS), but only for terminating the various streams of control characters that are interspersed between the data characters. Any character less than x'40' can also be used as the EOS, but it is simpler to stick to the standard x'AE'. The x'AF' control character specifies which character to use for the sign (first for positive or zero values and after the EOS for negative values) for fixed format pictures. The *EDIT* instruction works from left to right through the source and the edit mask. The x'AA' control character specifies that the corresponding source digit is to be copied to the receiver, even if we are still among the leading zeroes (*i.e.* have no numeric "significance" yet). The x'B3' control character specifies that the character that follows is to be inserted at the current position in the receiver.

Here is the next example:

```
DCL CON PICTURE2 CHAR(25) INIT(X' B14040AE60AEB2B2B06BAEB2B2B2B06BAEB2B2AAB34BAEAAAA' );  
EDIT OUT-DATA2, IN-DATA, PICTURE2;
```

Edit Mask Part	Explanation for --, ---, --9. 99
B1 40 40 AE 60 AE	Fill: <b>space</b> ; Positive: <b>space</b> ; Negative: <b>minus sign</b>
B2 B2	-- either a source digit or the <i>fill</i> character to be used (two of them)
B0 6B AE	, separator comma
B2 B2 B2	--- source digits (if significant) otherwise fill character
B0 6B AE	, separator comma
B2 B2 AA	--9 last of the three always source digit
B3 4B AE	. decimal point
AA AA	99 source decimals

The x'B1' control character specifies first which character to use as a fill character before the sign for floating formats; then follow the characters to use for the sign (first for positive or zero values and after the EOS for negative values). Note that the sign characters could actually be a string, *e.g.* CR for credit or DB for debit. The x'B0' control character specifies to insert either the character given (if significance has been reached) or the fill character. The x'B2' control character specifies to use either a source digit (if significance has been reached) or the fill character.

More examples:

```
DCL CON PICTURE3 CHAR(23) INIT(X' AA B36BAEAAAAA B36BAEAAAAA B34BAEAAAA AF40AE60AE ');
EDIT OUT-DATA3, IN-DATA, PICTURE3;
```

Edit Mask Part	Explanation for 9, 999, 999. 99-
AA	9 source digit
B3 6B AE	, separator comma
AA AA AA	999 source digits
B3 6B AE	, separator comma
AA AA AA	999 source digits
B3 4B AE	. decimal point
AA AA	99 source decimals
AF 40 AE 60 AE	Positive: <b>space</b> ; Negative: <b>minus sign</b>

```
DCL DD PICTURE4 CHAR(27) INIT(X' B140AEAE B2B06BAE B2B2B2B06BAE B2B2AA B34BAEAAAA AF40AE60AE ');
EDIT OUT-DATA4, IN-DATA, PICTURE4;
```

Edit Mask Part	Explanation for Z, ZZZ, ZZ9. 99-
B1 40 AE AE	Fill: <b>space</b> ; note that there are no sign characters, only EOSs
B2	Z either a source digit or the <i>fill</i> character to be used
B0 6B AE	, separator comma
B2 B2 B2	ZZZ source digits (if significant) otherwise fill character
B0 6B AE	, separator comma
B2 B2 AA	ZZ9 last of the three always source digit
B3 4B AE	. decimal point
AA AA	99 source decimals
AF 40 AE 60 AE	Positive: <b>space</b> ; Negative: <b>minus sign</b>

The starting x'B1' control sequence does not contain any sign character strings (as the picture specifies a trailing sign), but we still need two EOSs. Let's finish with two more examples:

```
02 OUT-DATA5 PIC 9B999B999. 99+. Result = 1 234 567. 89+ 0 000 001. 23+
```

```
DCL CON PICTURE5 CHAR(23) INIT(X' AA B340AEAAAAA B340AEAAAAA B34BAEAAAA AF4EAE60AE ');
EDIT OUT-DATA5, IN-DATA, PICTURE5;
```

Edit Mask Part	Explanation for 9B999B999. 99-
AA	9 source digit
B3 40 AE	ь separator blank
AA AA AA	999 source digits
B3 40 AE	ь separator blank
AA AA AA	999 source digits
B3 4B AE	. decimal point
AA AA	99 source decimals
AF 4E AE 60 AE	Positive: <b>plus sign</b> ; Negative: <b>minus sign</b>

```
02 OUT-DATA6 PIC *****9. 99-. Result = **1234567. 89- *****1. 23-
```

```
DCL CON PICTURE6 CHAR(23) INIT(X' B15CAEAE B2B2B2B2B2B2B2B2AA B34BAEAAAA AF40AE60AE ');
EDIT OUT-DATA6, IN-DATA, PICTURE6;
```

Edit Mask Part	Explanation for *****9. 99-
B1 5C AE AE	Fill: <b>asterisk</b> ; note that there are no sign characters, only EOSs
B2 B2 B2 B2 ... AA	***...9 fill with *, end with one source digit
AA AA AA	999 source digits
B3 4B AE	. decimal point
AA AA	99 source decimals
AF 4E AE 60 AE	Positive: <b>plus sign</b> ; Negative: <b>minus sign</b>

## Parameter Length Conformance

An *edit digit count* (x'0C04') exception is signaled if:

- The end of the source is reached and there are more control characters that correspond to digits in the edit mask.
- The end of the edit mask is reached and there are more digit positions in the source.

A *length conformance* (x'0C08') exception is signaled if:

- The end of the edit mask is reached and there are more control character positions in the receiver.
- The end of the receiver is reached and more positions remain in the edit mask.
- The number of B2s following a B1 cannot accommodate the longer of the two floating strings.

The moral here is simple: be careful with the lengths. The receiver cannot be too large either. Everything must just fit.

## Left-Justifying the Number

Numbers are usually right-justified in a field and the EDI T instruction is designed to produce right-justified results, i.e. values that “butt” up against the right-hand side of the field. Occasionally, one has the need to produce results that are *left-justified*. The following little MI-program, **MI EDTNBR**, shows how first to use the EDI T instruction to edit a numeric value and then how to use the VERI FY instruction to help left-justify the result:

```
DCL DD RECEIVER CHAR(13);
DCL DD NUMBER PKD(9,2) INIT(P' -1.23');
DCL DD PICTURE CHAR(25) /* COBOL PICTURE --,---,--9.99 */
    INIT(X' B14040AE60AEB2B2B06BAEB2B2B06BAEB2B2AAB34BAEAAAA');
/*          - - , - - - , - - 9 . 9 9 */

DCL SPCPTR .PARM1 PARM;
DCL DD PARM-NBR PKD(15,5) BAS(.PARM1);

DCL OL PARMS(.PARM1) PARM EXT MIN(1);

ENTRY * (PARMS) EXT;
  CPYINV      NUMBER, PARM-NBR;
  EDIT       RECEIVER, NUMBER, PICTURE;

  CPYBLAP     MSG-TEXT, "[", " ";
  CPYBLAP     MSG-TEXT(2:14), RECEIVER, " ";

DCL DD WHERE BIN(2);
DCL DD LENGTH BIN(2);

  VERI FY(B)  WHERE, RECEIVER, " "/ZER(=+3);
  SUBN       LENGTH, 14, WHERE; /* 14 = 13 + 1 */
  CPYBOLAP   RECEIVER, RECEIVER(WHERE:LENGTH), " ";

  CPYBLA     MSG-TEXT(18:1), "[";
  CPYBLAP    MSG-TEXT(19:14), RECEIVER, " ";
  CALLI     SHOW-MESSAGE, *, .SHOW-MESSAGE;

  RTX      *;

%INCLUDE SHOWMSG
```

Here is a sample run:

```
==> CALL MI EDTNBR PARM(-1.23)
From . . . LSVALGAARD 02/12/01 21:54:43
[ -1.23] [-1.23]
```

## Chapter 25

### Machine Indexes

#### ***What is a Machine Index?***

An *index* provides a mechanism to rapidly find an entry in a large table. Indexes are fundamental to many AS/400 components and are implemented below the MI-level, which means that there are MI-instructions to create, destroy, and use indexes. The low-level indexes are called *Machine Indexes*. Indexes are implemented as “partitioned binary radix trees”. Even if *how* they are implemented is really immaterial, it is of general interest to know the basic method behind the AS/400’s very efficient implementation of indexes.

#### **Binary Radix Trees.**

Consider the recurring problem of finding an entry in a list of numbers (“keys”). We store the numbers in a tree structure. The tree contains two types of nodes: branch nodes and leaf nodes. A branch node corresponds to a single bit in the number. Whether the bit is a zero or a one determines which of two lower-level nodes are selected as the next node to test. If we start at the top of the tree (the “root” - in the computer world, trees grow upside-down), the first node tests the first (leftmost) bit in the key. The second level has two branch nodes, one for the first bit being a zero and one for the first bit being a one. The third level has four nodes, the fourth eight, and so on.

If the key has twenty bits (about one million different values) there will be twenty levels. The bottom level will be leaf nodes that store the data associated with the keys. To find a leaf node we “descend” the branch nodes left or right depending on successive bits of the key. To minimize page faults, the tree is partitioned into “subtrees” each taking up a page, so that once we enter a page while searching for a particular entry, we stay on that page through all its relevant branch nodes. We thus exhaust the branch nodes in our path towards the leaf on the current page before continuing on the next page.

#### ***The Index Instructions***

There are seven instructions that work on indexes or as the MI-manual calls them: *independent* indexes. I know of no dependent indexes, so we’ll drop the “independent” prefix from now on. The instructions fall in two groups: index manipulation, and entry manipulation:

<b>CTRI NX</b>	Create Index
<b>MATI NXAT</b>	Materialize Index Attributes
<b>MODI NX</b>	Modify Index
<b>DESI NX</b>	Destroy Index
<b>INSI NXEN</b>	Insert Index Entry (also serves to update an entry)
<b>FNDI NXEN</b>	Find Index Entry
<b>RMVI NXEN</b>	Remove Index Entry

There are two variations of indexes, a most efficient type where the length of an entry is limited to 120 bytes (which is also the maximum size of the index key), and if you need it, an extended capability type with a maximum entry length of 2000 bytes. Indexes can have fixed-length or variable-length entries, be temporary or permanent, be inserted into a context (a library) or not.

#### **Security Level 40+ Considerations**

For some inexplicable (or at least unsatisfactorily justified) reason, IBM has chosen to disallow creation and destruction of indexes by user state programs when the system security level is 40 or above. You either have to promote (otherwise unnecessarily) your programs to system state or use the supplied (somewhat deficient) APIs. Just for the exercise, in the code that follows we’ll use both techniques. If direct creation fails, fall back to the API.



## The Search Engine Problem

The problem we'll use indexes to solve is that of making a simple *search engine*. The specifications for the search engine index are as follows:

- Read all the members in a source file
- For each member, read all lines (arbitrarily imposing an 80-character line length)
- For each line, extract all “words” ignoring “noise” words
- For each word, insert an entry in the index, where the key is the concatenation of the word, the member name, and the line number, and where the data is the text of the line
- If such an entry already exists, update it with the new data text

Each entry will be precisely 120 bytes long, so we can take advantage of the short form. Here is the layout of an entry (note that words are truncated to be no longer than 15 characters):

```
DCL DD THE-ENTRY CHAR(120);
  DCL DD THE-KEY      CHAR(40) DEF(THE-ENTRY) POS( 1);
    DCL DD THE-WORD    CHAR(25) DEF(THE-KEY)  POS( 1);
    DCL DD THE-MEMBER  CHAR(10) DEF(THE-KEY)  POS(26);
    DCL DD THE-LINE    ZND(5,0) DEF(THE-KEY)  POS(36);

  DCL DD THE-DATA      CHAR(80) DEF(THE-ENTRY) POS(41);
```

The search engine problem can be broken in to a number of sub-problems, that we can solve step by step:

1. Creating the index
2. Extracting the “words”
3. Inserting the word into the index
4. Searching the index for a word and showing all matches

The first sub-problem to solve, then, is creating the index.

## Index Object Subtypes

An index has type **x'0E'**. As you can see in Appendix B, there are many kinds in indexes, attesting to the importance of the machine index to OS/400. Even things like job queues (subtype **x'01'**), output queues (**x'02'**), and message files (**x'03'**) are implemented as indexes (despite being called “queues” or “files”). We have already met other indexes, such as the “Authorized User Table” (**x'E5'**) in Chapter 14, and the “Job Index” (**x'A4'**) in Chapter 9.

For the search engine index we'll be using a subtype of **x'0A'** for “user index” (**\*USRI DX**). As the name implies, a user index will not interfere with the management of OS/400. In light of this it is even more of a shame that creation of such an index is prohibited at security level 40 and above.

## Creating the Index

If the index already exists, we continue to the next step, otherwise we should create the index; so, first we check for its existence:

```
DCL SYSPTR .THE-IDX;

CHECK-IF-INDEX-EXISTS:
  CPYBLA      RESOLVE-TYPE, X'0401' ;          /* first resolve to library */
  CPYBLAP     RESOLVE-NAME, PARM-LIBRARY, " ";
  RSLVSP      .IDX-CONTEXT, RESOLVE, *, *;

  CPYBLA      RESOLVE-TYPE, X'0EOA' ;          /* then to index in that context */
  CPYBLAP     RESOLVE-NAME, PARM-INDEX, " ";
  RSLVSP      .THE-IDX, RESOLVE, .IDX-CONTEXT, *;
```

The following exception handler takes care of directing us to the code where we create the index:

```
DCL EXCM * EXCID(H'2201') BP(OBJECT-NOT-FOUND) CV(X'00000000') IMD;
```

The Create Index instruction takes two operands:

```
CRТИNX      .THE-IDX, .IDX-DESCR; /* CREATE INDEX, IF POSSIBLE */
```

The first operand is a system pointer to the index object that was created (if one was). This pointer will be used by all the other index instructions to identify to the index. The second operand is a space pointer to a template describing the index. You initialize fields in the template prior to creating the index.

Here is the structure of the index description template:

```
DCL DD      IDX-DESCR CHAR(200) BDRY(16); /* must be 16-byte aligned */
DCL DD      IDX-BYTES-PROV      BIN(4) DEF(IDX-DESCR) POS( 1);
DCL DD      IDX-BYTES-AVAIL     BIN(4) DEF(IDX-DESCR) POS( 5);
DCL DD      IDX-OBJ-TYPE        CHAR(1) DEF(IDX-DESCR) POS( 9);
DCL DD      IDX-OBJ-SUBTYPE      CHAR(1) DEF(IDX-DESCR) POS(10);
DCL DD      IDX-OBJ-NAME        CHAR(30) DEF(IDX-DESCR) POS(11);
DCL DD      IDX-CRT-OPTIONS      CHAR(4) DEF(IDX-DESCR) POS(41);
DCL DD      IDX-ASP-NUMBER       BIN(4) DEF(IDX-DESCR) POS(45);
DCL DD      IDX-SIZE-OF-SPACE    BIN(4) DEF(IDX-DESCR) POS(49);
DCL DD      IDX-INITIAL-VALUE    CHAR(1) DEF(IDX-DESCR) POS(53);
DCL DD      IDX-PERFM-OPTIONS    CHAR(4) DEF(IDX-DESCR) POS(54);
DCL DD      IDX-MBZ              CHAR(3) DEF(IDX-DESCR) POS(58);
DCL DD      IDX-EXT-OFFSET       BIN(4) DEF(IDX-DESCR) POS(61);
DCL SYSPTR . IDX-CONTEXT         DEF(IDX-DESCR) POS(65);
DCL SYSPTR . IDX-ACCESS-GROUP    DEF(IDX-DESCR) POS(81);
DCL DD      IDX-ATTRS            CHAR(1) DEF(IDX-DESCR) POS(97);
DCL DD      IDX-ENTRY-LENGTH     BIN(2) DEF(IDX-DESCR) POS(98);
DCL DD      IDX-KEY-LENGTH       BIN(2) DEF(IDX-DESCR) POS(100);

DCL DD      IDX-EXT              CHAR(64) DEF(IDX-DESCR) POS(129);
DCL DD      IDX-DOMAIN           CHAR(2) DEF(IDX-EXT)   POS(21);
```

Comments about each field:

Field		Comment
BYTES-PROV		Ignored
BYTES-AVAIL		Ignored
OBJ-TYPE		Ignored (set automatically to x'0E')
OBJ-SUBTYPE		We'll use x'0A' for *USRIDX)
OBJ-NAME		Remember to pad with spaces to 30 characters
CRT-OPTIONS		Creation options
	Bit 0	0=Temporary, 1=Permanent
	Bit 1	0=Fixed length space, 1=Variable length space
	Bit 2	0=Not placed in a context, 1=Placed in context given in . IDX-CONTEXT
	Bit 3	0=Not placed as member of an access group, 1=Created as member of access group given by . IDX-ACCESS-GROUP
	Bits 4-12	0, Reserved
	Bit 13	0=Initialize space, 1=Don't initialize space
	Bits 14-19	0, Reserved
	Bit 20	For index object: 0=No protection at low SECLVLs, 1=Protected at all times
	Bit 21	As for Bit 20, but for the associated space (if any)
	Bits 22-31	0, Reserved
ASP-NUMBER		0=Use System ASP, 2-16=Use that ASP
SIZE-OF-SPACE		Size of associated space (if any)
INITIAL-VALUE		Initial value of space (if bit 13 is set in CRT-OPTIONS)
PERFM-OPTIONS		Performance options
	Bit 0	0=No special alignment of space, 1=Aligned of 512-byte boundary
	Bits 1-4	0, Reserved
	Bit 5	0=Process storage pool used, 1=Machine pool used
	Bit 6	0, Reserved

	Bit 7	0=Minimum transfer size, 1=Machine transfer size
	Bits 8-31	0, Reserved
MBZ		0, Reserved. All three bytes <b>Must Be Zeroes</b>
EXT-OFFSET		Offset to Extension area later on in the template
.IDX-CONTEXT		System pointer to context holding index (if any)
.IDX-ACCESS-GROUP		System pointer to access group (if any)
ATTRS		Attributes
	Bit 0	0=Fixed-length entries, 1=Variable-length entries
	Bit 1	0=No immediate update, 1=Immediate update
	Bit 2	0=No insertion by key, 1=Insertion by key
	Bit 3	0=Entries do not contain pointers, 1=Entries can contain pointers
	Bit 4	0=Optimize for random access, 1=Optimize for sequential access
	Bit 5	0=Max. entry length 120 bytes, 1=Max. entry length 2000 bytes
	Bits 6-7	0, Reserved
ENTRY-LENGTH		Length of entry, if fixed-length entries are specified
KEY-LENGTH		Length of key, if insertion by key is specified

The extension offset (if non-zero) specifies the byte offset from the beginning of the template to the template extension. This offset must be a multiple of 16. In our example, we use 128. The extension has this format:

Field	Comment
20 bytes	0, Reserved
DOMAIN	x'0000'=Chosen by the system, x'0001'=User
42 bytes	0, Reserved

With all this in mind, we now initialize the template as follows:

```

OBJECT-NOT-FOUND:
CPYBLAP      IDX-OBJ-NAME, PARM-INDEX, " ";
CPYBLA       IDX-OBJ-TYPE,      X'OE';          /* INDEX                */
CPYBLA       IDX-OBJ-SUBTYPE,    X'OA';          /* USER INDEX           */
CPYBLA       IDX-CRT-OPTIONS,    X'A0040000';   /* PERMANENT, IN CTX    */
CPYBLA       IDX-PERFM-OPTIONS,  X'00000000';   /* DEFAULTS             */
CPYBLA       IDX-MBZ,           X'000000';      /* MUST BE ZEROES       */
CPYBLA       IDX-ATTRS,         X'20';          /* FIXED, KEYED, SHORT */
CPYNV        IDX-ASP-NUMBER,     0;              /* USE SYSTEM ASP       */
CPYNV        IDX-SIZE-OF-SPACE,  0;              /* INITIAL SPACE SIZE   */
CPYNV        IDX-ENTRY-LENGTH,  120;             /* KEY + DATA          */
CPYNV        IDX-KEY-LENGTH,     40;             /* WORD + SEQ-NBR       */
CPYNV        IDX-EXT-OFFSET,     128;            /* EXTENSION            */
CPYBREP      IDX-EXT,           X'00';
CPYBLA       IDX-DOMAIN, X'0001'; /* USER DOMAIN */

CRTINXB      .THE-IDX, .IDX-DESCR; /* CREATE INDEX, IF POSSIBLE */
              OPEN-INPUT-FILE;

```

Finally, we attempted to create the index and go to process the input file member. Should we fail (with MCH6801) because of protection violation, we must use the "Create User Index" API, **QUSCRTUI**:

```

DCL EXCM * EXCID(H'4401') BP(CREATE-INDEX-API) CV(X'00000000') IMD;
CREATE-INDEX-API:
CPYBLA       UI-NAME,           PARM-INDEX;
CPYBLA       UI-LIBRARY,        PARM-LIBRARY;
CPYNV        UI-ENTRY-LENGTH,  120;
CPYNV        UI-KEY-LENGTH,     40;
CALLX        .SEPT(5389), QUSCRTUI, *;
RSLVSP      .THE-IDX, RESOLVE, .IDX-CONTEXT, *;
B           OPEN-INPUT-FILE;

```

Note that with the API we are limited to 10-character library and index names. With our knowledge of the creation template, we can easily understand the parameters of the API (I've omitted the declarations of the space pointers to each of these):

```
DCL DD UI -NAME-LI BRARY CHAR(20); /* 10-char name, 10-char library */
DCL DD UI -ATTR CHAR(10) INIT("INDEX"); /* ARBITRARY ATTR */
DCL DD UI -TYPE CHAR(1) INIT("F"); /* FIXED LENGTH */
DCL DD UI -ENTRY-LENGTH BIN(4);
DCL DD UI -KEYED CHAR(1) INIT("1"); /* INSERT KEYED */
DCL DD UI -KEY-LENGTH BIN(4);
DCL DD UI -UPDATE CHAR(1) INIT("0"); /* NO IMMEDIATE UPDATE */
DCL DD UI -ACCESS CHAR(1) INIT("0"); /* OPTIMIZE RANDOM ACCESS */
DCL DD UI -AUTHORITY CHAR(10) INIT("*ALL"); /* ALL HAVE ACCESS */
DCL DD UI -DESCRIPTION CHAR(50) INIT("My Index");
DCL DD UI -REPLACE CHAR(10) INIT("*YES"); /* REPLACE IF EXISTS */
DCL DD UI -ERROR BIN(4) INIT(0); /* PERCOLATE ERROR */

DCL OL QUSCRTUI (. UI -NAME-LI BRARY, . UI -ATTR, . UI -TYPE, . UI -ENTRY-LENGTH,
. UI -KEYED, . UI -KEY-LENGTH, . UI -UPDATE, . UI -ACCESS,
. UI -AUTHORITY, . UI -DESCRIPTION, . UI -REPLACE, . UI -ERROR) ARG;
```

A significant difference between the two ways of creating an index is that the QUSCRTUI , API throws in a gratuitous associated space, while the **CRTIDX**-instruction does not.

## Reading the File Member

To make the program a bit more general, we'll delegate the job of finding members of the file to the caller of our program. So, given the library, file, and member names, opening the member is *vieux chapeau* (see Chapter 12 for a refresher):

```
DCL SPCPTR .ODP;
DCL SPC ODP BAS(.ODP);
DCL DD ODP.DCB BIN(4) DEF(ODP) POS(17);

DCL SPCPTR .DCB;
DCL SPC DCB BAS(.DCB);
DCL DD DCB-GET BIN(2) DEF(DCB) POS(25);
DCL DD DCB-PUT BIN(2) DEF(DCB) POS(33);

DCL SPCPTR .IFCB INIT(IFCB);
DCL DD IFCB CHAR(214) BDRY(16);
DCL SPCPTR .IFCB-ODP DEF(IFCB) POS( 1);
DCL SPCPTR .IFCB-INBUF DEF(IFCB) POS( 17);
DCL DD IFCB-FILE CHAR(10) DEF(IFCB) POS(129);
DCL DD IFCB-LIB-ID BIN(2) DEF(IFCB) POS(139) INIT(72);
DCL DD IFCB-LIBRARY CHAR(10) DEF(IFCB) POS(141);
DCL DD IFCB-MBR-ID BIN(2) DEF(IFCB) POS(151) INIT(73);
DCL DD IFCB-MEMBER CHAR(10) DEF(IFCB) POS(153);
DCL DD IFCB-FLAGS-1 CHAR(1) DEF(IFCB) POS(175) INIT(X'80');
DCL DD IFCB-FLAGS-2 CHAR(1) DEF(IFCB) POS(176) INIT(X'20');
DCL DD IFCB-LENGTH-ID BIN(2) DEF(IFCB) POS(209) INIT(1);
DCL DD IFCB-RECORD-LENGTH BIN(2) DEF(IFCB) POS(211) INIT(92);
DCL DD IFCB-NO-MORE-PARMS BIN(2) DEF(IFCB) POS(213) INIT(32767);

DCL OL OPEN-I(.IFCB) ARG;
DCL OL CLOSE-I(.IFCB) ARG;

DCL CON OPEN-ENTRY BIN(2) INIT(12);
DCL CON CLOSE-ENTRY BIN(2) INIT(11);
DCL DD GET-ENTRY BIN(2);

DCL DD INBUF CHAR(92) BAS(.IFCB-INBUF);
DCL DD INDATA CHAR(80) DEF(INBUF) POS(13);

DCL DD GET-OPTION BIN(4) INIT(H'03000001');
DCL SPCPTR .GET-OPTION INIT(GET-OPTION);
DCL OL GET-OPERATION(.IFCB, .GET-OPTION, .NULL);

OPEN-INPUT-FILE:
CPYBLA IFCB-MEMBER, PARM-IN-MEMBER;
CPYBLA IFCB-FILE, PARM-IN-FILE;
CPYBLA IFCB-LIBRARY, PARM-IN-LIBRARY;
CALLX .SEPT(OPEN-ENTRY), OPEN-I, *;

CPYBWP .ODP, .IFCB-ODP;
ADDSP .DCB, .ODP, ODP.DCB;
CPYNV GET-ENTRY, DCB-GET;
```

```

TRANSFORM-FILE:
  CALLX      .SEPT(GET-ENTRY), GET-OPERATION, *;
  B          TRANSFORM-FILE;

```

At TRANSFORM-FILE, the above code does nothing except read the file to the end. Our next sub-problem is to actually extract the words from each record read.

## Extracting Words

First we must define what we mean by a “word”. Note that our specifications also used quotes. Defining words is not our main concern here, so we’ll take a simple (but, as it turns out, quite adequate) approach:

- First append a space to the end of the record
- Then translate any of the following characters to a space: `()[]{}:;,”?/><!%^+=\`
- A word is now something that ends at a space
- If the last character of the word is a period, drop the period from the word
- If the word at this point has a length of less than two and is not a digit, skip it

The following straightforward code implements the above algorithm:

```

TRANSFORM-FILE:
  CALLX      .SEPT(GET-ENTRY), GET-OPERATION, *; /* AT END: EOF */
  ADDN(S)    REC-NBR, 1;
  CPYBLA     THE-RECORD(81:2), " ."; /* SENTINELS */
  XLATE      THE-RECORD(1:80), INDATA, 'O[]{}:;, "/><!%^+=\';
  CPYNV      FROM, 0;
EXTRACT-AND-PROCESS-WORD:
  CALLI      GET-A-WORD, *, .GET-A-WORD; /*AT END: TRANSFORM FILE*/
  [here we must do something with the word]
  B          EXTRACT-AND-PROCESS-WORD;

DCL EXCM * EXCID(H' 5001') BP(EOF) CV("CPF") IMD;
EOF:
  CALLX      .SEPT(CLOSE-ENTRY), CLOSE-I, *;

DCL INSPTR .GET-A-WORD;
ENTRY       GET-A-WORD INT;
START-A-WORD:
  CPYBREP    A-WORD, " ";
  CPYNV      TO, 0;
GET-FIRST-CHAR:
  ADDN(S)    FROM, 1;
  CMPBLA(B)  THE-RECORD(FROM:1), " "/EQ(GET-FIRST-CHAR);
  CMPNV(B)   FROM, 80/HI(TRANSFORM-FILE);

```

Because our *sentinel* at the end of THE-RECORD is a space followed by a non-space, we’ll eventually find the non-space and can then ask if we are off the end (FROM > 80). If not, we continue:

```

ADD-CHAR-TO-WORD:
  ADDN(S)    TO, 1;
  CPYBLA     A-WORD(TO:1), THE-RECORD(FROM:1);
  ADDN(S)    FROM, 1;
  CMPBLA(B)  THE-RECORD(FROM:1), " "/NEQ(ADD-CHAR-TO-WORD);

  CMPBLA(B)  A-WORD(TO:1), ". "/NEQ(=+3); /* period at end? */
  CPYBLA     A-WORD(TO:1), ". ";
  SUBN(SB)   TO, 1/ZER(START-A-WORD);

  CMPNV(B)   TO, 1/HI(=+2);
  VERIFY(B)  WHERE, A-WORD(1:1), "0123456789"/POS(START-A-WORD);

CHECK-FOR-NOISE-WORD:
  SCAN(B)    WHERE, NOISE-WORDS, A-WORD(1:TO)/POS(START-A-WORD);
  B          .GET-A-WORD;

DCL DD NOISE-WORDS CHAR(128);
DCL DD *      (2) CHAR(64) DEF(NOISE-WORDS) POS(1) INIT(
  "DD DCL POS INIT DEF CHAR BIN SPCPTR SYSPTR PARM ENTRY INT EXT "
  "INSPTR BAS                                     ");

DCL DD A-WORD CHAR(81); /* later on we'll use the first 25 chars */
DCL DD WHERE BIN(4);

```

```
DCL DD FROM    BIN(2);
DCL DD TO      BIN(2);
```

The noise words I have chosen are from the viewpoint of searching MI source code. A more fancy approach would have been to have read the noise words from a file or even to check in another index [!] of noise words. But, as said, the present chapter is not an exercise in word extraction. With a word in our hand, we now turn to our third sub-problem: inserting the word into the index.

## Inserting an Entry

One can insert several (up to 4095) entries at the same time. Not much is gained in speed as the page-splitting will have to be done in the same way, regardless of the number of entries inserted, so we'll not buffer up words to be inserted, but shall simply insert the word entries one at a time as we get them:

```
EXTRACT-AND-PROCESS-WORD:
CALLI      GET-A-WORD, *, .GET-A-WORD; /*AT END: TRANSFORM-FILE*/
CALLI      PUT-A-WORD, *, .PUT-A-WORD;
B          EXTRACT-AND-PROCESS-WORD;
```

The **INSINXN** instruction takes three operands; the first operand is the system pointer to the index, the second is a space pointer to a list of entries to insert:

```
DCL SPCPTR .ENTRIES INIT(ENTRIES);
DCL DD     ENTRIES(1) CHAR(120);
DCL DD     THE-ENTRY CHAR(120) DEF(ENTRIES) POS(1);
DCL DD     THE-KEY CHAR(40) DEF(THE-ENTRY) POS( 1);
DCL DD     THE-WORD CHAR(25) DEF(THE-KEY) POS( 1);
DCL DD     THE-MEMBER CHAR(10) DEF(THE-KEY) POS(26);
DCL DD     THE-LINE CHAR(05) DEF(THE-KEY) POS(36);
DCL DD     THE-DATA CHAR(80) DEF(THE-ENTRY) POS(41);
```

Note that indexes do not need to be opened or closed. They are generally sharable. Each insert operation is atomic meaning all entries in the list are inserted together. This is one reason for the limit of 4095 entries at one time: we don't want an index operation to monopolize the system

The third operand specifies what to do with the entry:

```
DCL SPCPTR .THE-OPTION INIT(THE-OPTION);
DCL DD     THE-OPTION CHAR(14);
DCL DD     RULE-OPTION CHAR(2) DEF(THE-OPTION) POS( 1);
DCL DD     DATA-LENGTH BIN(2) DEF(THE-OPTION) POS( 3);
DCL DD     DATA-OFFSET BIN(2) DEF(THE-OPTION) POS( 5);
DCL DD     WANTED-COUNT BIN(2) DEF(THE-OPTION) POS( 7);
DCL DD     RETURN-COUNT BIN(2) DEF(THE-OPTION) POS( 9);
DCL DD     ENTRY-LENGTH BIN(2) DEF(THE-OPTION) POS(11) INIT(120);
DCL DD     ENTRY-OFFSET BIN(2) DEF(THE-OPTION) POS(13) INIT(0);
```

The **RULE-OPTION** can be one of:

- x'0001' Insert entry with unique value (Exception MCH2401 if entry already there)
- x'0002' Insert entry with replacing data portion if key already there
- x'0003' Insert entry only if key not already present (MCH2401 if key already present)

Value x'0001' is for indexes without keys, and the other ones are for indexes *with* keys.

The **DATA-LENGTH** and **-OFFSET** are ignored for insert operations, as the values are taken from the entry list. **WANTED-COUNT** must be set to the number of entries in the list, and **RETURN-COUNT** will be returned with the number of entries actually inserted.

The options end with a list of pairs of numbers; the first of the pair is length of the item, the second is an offset to the entry within the **ENTRIES** area. For the first (and in our case, only) entry, the offset is the number of bytes from the beginning of area to the first byte of the first entry. For any succeeding entry, the offset is the number of bytes from the beginning of the immediately preceding entry to the first byte of the entry.

We can now code the **PUT-A-WORD** routine:

```
DCL INSPTR .PUT-A-WORD;
```

```

ENTRY      PUT-A-WORD INT;
CPYBLA     THE-WORD, A-WORD; /* TRUNCATE */
CPYBLA     THE-MEMBER, IFCB-MEMBER;
CPYBRA     THE-LINE, REC-NBR;
CPYBLA     THE-DATA, INDATA;

CPYBLA     RULE-OPTION, X'0002'; /* INSERT OR UPDATE */
CPYNV      WANTED-COUNT, 1; /* INSERT ONE ENTRY */
INSIXEN    THE-IDX, ENTRIES, THE-OPTION;
B          PUT-A-WORD;

```

## Internal Structure of an Index

I ran the **MI KWDI DX** program, that results from putting all the pieces described above together, on a member with the text shown in the left-hand column below:

1	The text consists of a sequence of lines with one word per line. A word is formed by
12	adding one new character to the end of the previous word. A property of the binary
123	radix tree is that since the branch nodes reflect bits of the key, these bits need not be
1234	stored in the leaf nodes as they can be reconstructed from the path taken along the
...	branch nodes. The first page of the object holds the usual generic object information;
123456789	the data starts at the next page at offset x'1000'.

Apart from the first entry on the page that has extra information related to page management, the “overhead” for each entry is only 9 bytes, which I have highlighted below. You can also see how leading bytes of the key that are the same from entry to entry are not stored:

Address	03A2C0B22A 000000			
000000	00010020 00900001	03A2C0B2 2A000000	..... °... s{¥....	
000010	00010000 00000000	00000000 FF000000	.....	← no primary space
000020	80000E0A E3C5E2E3	C9C4E740 40404040	0... TESTI DX	← types and name
000030	40404040 40404040	40404040 40404040		
000100	24000078 00280000	0000000E 00000000	...}.....	← data, key lengths
000110	00000000 00000000	C0000000 00000000	..... {.....	
000120	03A2C0B2 2A000400	00000000 00000000	.. s{¥.....	
000130	00000000 00000000	00000000 00000000	.....	
000140	00000000 00000000	00000000 00000000	.....	
000400	00000000 EAEA0401	D4C1C3C8 C9D5C4E7	.... 22... MACHI NDX	
000410	00000000 0000000E	00000000 0000000E	.....	
000470	00000000 00000000	00000000 00000000	.....	
000480	03A2C0B2 2A001000	00000000 00000001	.. s{¥.....	← start of data
000490	88000000 00000000	00000000 00000001	h.....	
0004A0	00000800 00002000	03A2C0B2 2A002000	..... s{¥....	
0004B0	00000000 00000000	00000000 00000000	.....	
0004C0	00000000 00000000	CECBCC53 F4000000	..... 00004...	← secondary space?
0004D0	00000000 0000000E	00000000 08000F00	.....	
001000	940559CC 093C16C4	60000088 05DFF140	m. Bö... D... h. y1	
001010	40404040 40404040	40404040 40404040		
001020	40404040 404040D2	C5E8E240 40404040	KEYS	
001030	40F0F0F0 F0F1F140	40404040 40404040	000011	
001040	40404040 40404040	40404040 40404040		
001050	40404040 40404040	40404040 40404040		
001060	40404040 40404040	40404040 40404040		
001070	40404040 40404040	40404040 40404040		
001080	40404040 40407510	0F88010D 00404040		
0010A0	40404040 404040D2	C5E8E240 40404040	KEYS	
0010B0	40F0F0F0 F0F2F1F2	40404040 40404040	0000212	
0010C0	40404040 40404040	40404040 40404040		
0010D0	40404040 40404040	40404040 40404040		
0010E0	40404040 40404040	40404040 40404040		
0010F0	40404040 40404040	40404040 40404040		
001100	40404040 40407510	9088010C 00108FF3	í. °h... ±3	
001110	40404040 40404040	40404040 40404040		
001120	40404040 4040D2C5	E8E24040 40404040	KEYS	
001130	F0F0F0F0 F3F1F2F3	40404040 40404040	00003123	
001140	40404040 40404040	40404040 40404040		
001150	40404040 40404040	40404040 40404040		
001160	40404040 40404040	40404040 40404040		
001170	40404040 40404040	40404040 40404040		
001180	40404040 40741110	88030A00 110FF440	È. h... 4	
001190	40404040 40404040	40404040 40404040		

0011A0	40404040	D2C5E8E2	40404040	4040F0F0	KEYS	00
0011B0	F0F0F4F1	F2F3F440	40404040	40404040	0041234	
0011C0	40404040	40404040	40404040	40404040		
0011D0	40404040	40404040	40404040	40404040		
0011E0	40404040	40404040	40404040	40404040		
0011F0	40404040	40404040	40404040	40404040		
001200	40404073	118F8803	0800118E	F5404040	Ê· ±h· . . . þ5	
001210	40404040	40404040	40404040	40404040		
001220	40D2C5E8	E2404040	404040F0	F0F0F0F5	KEYS	00005
001230	F1F2F3F4	F5404040	40404040	40404040	12345	
001240	40404040	40404040	40404040	40404040		
001250	40404040	40404040	40404040	40404040		
001260	40404040	40404040	40404040	40404040		
001270	40404040	40404040	40404040	40404040		
001280	72120D88	00FA0012	0CF64040	40404040	Ê· · h· ¢· . . . 6	
001290	40404040	40404040	40404040	40D2C5E8	KEY	
0012A0	E2404040	404040F0	F0F0F0F6	F1F2F3F4	S	000061234
0012B0	F5F64040	40404040	40404040	40404040	56	
0012C0	40404040	40404040	40404040	40404040		
0012D0	40404040	40404040	40404040	40404040		
0012E0	40404040	40404040	40404040	40404040		
0012F0	40404040	40404040	40404040	77128A88	Ê· «h	
001300	01F40012	89F74040	40404040	40404040	· 4· · i 7	
001310	40404040	40404040	D2C5E8E2	40404040	KEYS	
001320	4040F0F0	F0F0F7F1	F2F3F4F5	F6F74040	000071234567	
001330	40404040	40404040	40404040	40404040		
001340	40404040	40404040	40404040	40404040		
001350	40404040	40404040	40404040	40404040		
001360	40404040	40404040	40404040	40404040		
001370	40404040	40404070	13068801	0E001305	ø· · h· . . . .	
001380	F8404040	40404040	40404040	40404040	8	
001390	4040D2C5	E8E24040	40404040	F0F0F0F0	KEYS	0000
0013A0	F8F1F2F3	F4F5F6F7	F8404040	40404040	812345678	
0013B0	40404040	40404040	40404040	40404040		
...						
001620	40404040	40404040	40404040	40404040		
001630	40404040	40404040	40404040	40404040		
001640	40404040	40404040	40404040	40405C15	%·	
001650	E16C1657	0015E0C3	40404040	40404040	· %· ÿ· · \C	
001660	40404040	40D2C5E8	E2404040	404040F0	KEYS	0
001670	F0F0F1F4	F9F8F7F6	F5F4F3F2	F1C1C2C3	0014987654321ABC	
001680	40404040	40404040	40404040	40404040		
001690	40404040	40404040	40404040	40404040		
0016A0	40404040	40404040	40404040	40404040		
0016B0	40404040	40404040	40404040	40404040		
0016C0	40404040	00000000	00000000	00000000	· · · · · · · · · ·	
0016D0	00000000	00000000	00000000	00000000	· · · · · · · · · ·	
0016E0	00000000	00000000	00000000	00000000	· · · · · · · · · ·	
0016F0	00000000	00000000	00000000	00000000	· · · · · · · · · ·	

76	100F	88010D	00100E	The control information for each entry seems to consist of a byte holding the size of the part of the entry stored minus the size of the control information and the first character. The last 24 bits hold the offset from the beginning of the object to the <i>previous</i> entry. Deciphering the remaining control bits we leave as an exercise for the student ☺
75	1090	88010C	00108F	
74	1110	88030A	00110F	
73	118F	880308	00118E	
72	120D	8800FA	00120C	
71	128A	8801F4	001289	

All in all, the structure is remarkably compact and efficient.

### Iterating over All Members of a File

We had written **MI KWDI DX** to index words contained in a single member of a database file. Getting a list of members of a file is best done using a standard OS/400 API: “List Members”, **QUSLMBR**. Unfortunately, the API returns the list in a *user space*, so now we are faced with the problem of creating and accessing a user space. Just as with creating a user index, IBM has disallowed creation of user spaces by user-state programs, so we need to use yet another API “Create User Space”, **QUSCRTUS**, to create the space and one to access the space: “Get Pointer to User Space”, **QUSPTRUS**. The only saving grace of this mumbo-jumbo is that the list API allows listing based upon a *generic member* name, like “MI\*”.

The **MI FI LI DX** program starts out by creating a use space called “MI FI LI DX” in the QTEMP library:

```

FIND-MEMBERS:
  CPYBLAP      ML-US-SPACE  , "MI FI LI DX", " ";

```



```

CPYBLAP      ML-US-LI BRARY, "OTEMP"      , " ";
CALLX        .SEPT(5391), QUSCRTUS, *, /* CREATE USER SPACE */
CALLX        .SEPT(5396), QUSPTRUS, *, /* GET POINTER TO SPACE */

CPYBLA       ML-DB-FILE      , PARM-IN-FILE;
CPYBLA       ML-DB-LI BRARY, PARM-IN-LI BRARY;
CPYBLA       ML-MEMBERS     , PARM-IN-MEMBER; /* E.G. GENERIC: MI * */
CALLX        .SEPT(5093), QUSLMBR , *;

```

You can see the various parameter lists in the MI-program stored in the PROGRAMS area. The SEPT numbers for the APIs can be found in Appendix D.

```

DCL SPCPTR .US-SPACE-PTR INIT(US-SPACE-PTR);
DCL DD     US-SPACE-PTR CHAR(16) BDRY(16);
DCL SPCPTR .US-SPACE DEF(US-SPACE-PTR) POS(1); /* returned by QUSPTRUS */

DCL DD MEMBER CHAR(10) BAS(.US-SPACE);
DCL SYSPTR .MI KWDX;
DCL OL     MI KWDX (.US-SPACE, .PARM2, .PARM3, .PARM4, .PARM5) ARG;

```

The member list starts 512 bytes into the space, so we set up a pointer to the first member name:

```

ADDSP      .US-SPACE, .US-SPACE, 512; /* POINT TO MEMBER LIST */

NEXT-MEMBER:
CMPBLA(B)  MEMBER, " "/NHI(DONE);
CALLX      .MI KWDX, MI KWDX, *;
ADDSP      .US-SPACE, .US-SPACE, 10;
B          NEXT-MEMBER;

```

As long as a member name is present (its name is greater than spaces), we call **MI KWDX** with that member. The **ADDSP** instruction adds the binary value of its third operand to the offset portion of the space pointer given as the second operand and stores the result in the first operand. As member names are 10 characters long, by adding 10, you “walk” the list. When we are done with all the members, we want to send a message showing the size of the index in terms of entries:

```

DONE:
CALLI      SHOW-STATISTICS, *, .SHOW-STATISTICS;
RTX        *;

DCL SYSPTR .LIB;
DCL SYSPTR .IDX;

DCL SPCPTR .IDX-ATTR INIT(IDX-ATTR);
DCL DD     IDX-ATTR CHAR(113) BDRY(16);
DCL DD     IDX-BYTES-PROV BIN(4) DEF(IDX-ATTR) POS( 1) INIT(113);
DCL DD     IDX-BYTES-AVAIL BIN(4) DEF(IDX-ATTR) POS( 5);
DCL DD     IDX-TYPES-NAME CHAR(32) DEF(IDX-ATTR) POS( 9);
DCL DD     IDX-CRT-ATTRS CHAR(4) DEF(IDX-ATTR) POS(41);
DCL DD     IDX-SIZE-OF-SPACE BIN(4) DEF(IDX-ATTR) POS(49);
DCL DD     IDX-INSERTS UNSGND BIN(4) DEF(IDX-ATTR) POS(102);
DCL DD     IDX-REMOVES UNSGND BIN(4) DEF(IDX-ATTR) POS(106);
DCL DD     IDX-FINDS UNSGND BIN(4) DEF(IDX-ATTR) POS(110);

DCL DD ENTRIES-INSERTED ZND(10,0);
DCL DD ENTRIES-REMOVED ZND(10,0);
DCL DD ENTRIES-TOTAL ZND(10,0);

DCL INSPTR .SHOW-STATISTICS;
ENTRY      SHOW-STATISTICS INT;
CPYBLA     RESOLVE-TYPE, X'0401';
CPYBLAP    RESOLVE-NAME, PARM-LI BRARY, " ";
RSLVSP     .LIB, RESOLVE, *, *; /* resolve to library

CPYBLA     RESOLVE-TYPE, X'0EOA';
CPYBLAP    RESOLVE-NAME, PARM-I NDEX, " ";
RSLVSP     .IDX, RESOLVE, .LIB, *; /* resolve to index

MATINXAT   .IDX-ATTR, .IDX; /* materialize index attributes
CPYNV      ENTRIES-INSERTED, IDX-INSERTS;
CPYNV      ENTRIES-REMOVED , IDX-REMOVES;

```

The number of entries still present in the index is the difference between the number of entries inserted and the number of entries removed as returned by the “Materialize Index Attributes” instruction:

```

SUBN          ENTRIES-TOTAL      ,  IDX-INSERTS,  IDX-REMOVES;
CPYBLAP       MSG-TEXT,
"Inserted=1234567890, Removed=1234567890, Total=1234567890", " ";
/* 12345678[0123456789]123456789[1234567890]2345678[0123456789] */
CPYBLA       MSG-TEXT(10: 10), ENTRIES-INSERTED;
CPYBLA       MSG-TEXT(31: 10), ENTRIES-REMOVED;
CPYBLA       MSG-TEXT(50: 10), ENTRIES-TOTAL;
CALLI        SHOW-MESSAGE, *, . SHOW-MESSAGE;
B            . SHOW-STATISTICS;

```

We are using our trusted message routine to show the vital statistics of the index.

## Insertion Speed

Running the program:

```
===> CALL MI FILIDX PARM('MI*' QMI SRC LSVALGAARD QMI SRC LSVALGAARD)
```

builds an index, **QMI SRC**, containing all words from all members whose name starts with 'MI' from the file **QMI SRC**.

Timing the insertion of several thousand entries, we find (on a model 150) that insertion into the index takes place at a speed of more than 1700 entries per second. This is quite a respectable speed, even taking into account that most words are present several times over, so the time quoted includes all the times where the word was not inserted because the entry already existed. Since this situation almost always is what you meet in practice, simply dividing the total time by the number of resulting, unique entries gives the useful measure for this type of application.

## Searching the Index

Building an index is a fine exercise, but we really want to do something with it. Something like the interactive search screen shown here:

Search Word In Index				
Word to find . . . . .	<b>MATMATR</b>		Generic*	
Index . . . . .	<b>QMI SRC</b>		Name	
Library . . . . .			Name, *LIBL, blank	
Word	Member	Line	Text--->	
MATMATR	MI AUTREF	00152	MATMATR	. MCH-ATTR, X'0100'
MATMATR	MI ETPMON	00354	MATMATR	. MACHINE-ATTR, X'0
MATMATR	MI GETETP	00412	MATMATR	. MACHINE-ATTR, X'0
MATMATR	MI GETETP	00419	MATMATR	. MACHINE-ATTR, X'0
MATMATR	MI MCHATR	00024	MATMATR	. MACHINE-ATTR, MATM
MATMATR	MI MCHATR	00029	MATMATR	. MACHINE-ATTR, MATM
MATMATR	MI MCHATR	00034	MATMATR	. MACHINE-ATTR, MATM
MATMATR	MI MCHAT1	00026	MATMATR	. MACHINE-ATTR, MATM
MATMATR	MI MCHAT1	00031	MATMATR	. MACHINE-ATTR, MATM
MATMATR	MI MCHAT1	00036	MATMATR	. MACHINE-ATTR, MATM
Bottom				
F3=Exit    F11=Right/Left    F12=Cancel    F17=Top    F18=Bottom				

You key in a word (or a generic word ending in an asterisk), supply an index name (and a library if needed) and press the Enter key. The program retrieves the entries that match and presents them as scrollable list, showing the word, the member and the line whence it came and the left-most part of the text on that line. Press a function key to toggle between the left half and the right half of the line. The **MI SHWIDX** program does just that. The program uses the screen handler from Chapter 17. Here is part of the screen description:

```

DCL SPCPTR .SCREEN INIT(SCREEN);
DCL DD SCREEN CHAR(10) INIT("T010310020"); DCL DD S-TITLE CHAR(20) INIT("Search Word in Index");
DCL DD * CHAR(10) INIT("L010520000");

DCL DD * CHAR(10) INIT("L030010030"); DCL DD * CHAR(30) INIT("Word to find . . . . .");
DCL DD A-WORD CHAR(10) INIT("Q030320026"); DCL DD S-WORD CHAR(26) INIT("

```

```

DCL DD * CHAR(10) INIT("L030590015");          DCL DD *          CHAR(15) INIT(" Generi c*");
...
DCL DD * CHAR(10) INIT("L090010025");          DCL DD S-WORD-1          CHAR(25);
DCL DD * CHAR(10) INIT("L090270010");          DCL DD S-MEMBER-1      CHAR(10);
DCL DD * CHAR(10) INIT("L090380006");          DCL DD S-LINE-1        CHAR(06);
DCL DD * CHAR(10) INIT("N090450034");          DCL DD S-TEXT-1        CHAR(34);
DCL DD * CHAR(10) INIT("L090800000");          /*          SIZE 75 */

DCL DD * CHAR(10) INIT("L100010025");          DCL DD *          CHAR(25);
DCL DD * CHAR(10) INIT("L100270010");          DCL DD *          CHAR(10);
DCL DD * CHAR(10) INIT("L100380006");          DCL DD *          CHAR(06);
DCL DD * CHAR(10) INIT("N100450034");          DCL DD *          CHAR(34);
DCL DD * CHAR(10) INIT("L100800000");

```

The pattern is now repeated for several rows (SCREEN-LINES = 13).

The columns are defined as arrays, where the Array Element Offset (AEO) is given by the total size of the data items on one row:

```

DCL DD S-WORDS   (13)   CHAR(25) DEF(S-WORD-1 ) POS(1) AEO(125); /* 50 + 75 */
DCL DD S-MEMBERS (13)   CHAR(10) DEF(S-MEMBER-1) POS(1) AEO(125);
DCL DD S-LINES   (13)   CHAR(06) DEF(S-LINE-1 ) POS(1) AEO(125);
DCL DD S-TEXTS   (13)   CHAR(34) DEF(S-TEXT-1 ) POS(1) AEO(125);

```

The program is simple enough: resolve to the screen handler, open and show the screen, wait for operator input, then branch on the AID-key pressed::

```

ENTRY * EXT;
  CPYBLA          RESOLVE-TYPE, X' 0201' ;
  CPYBLAP         RESOLVE-NAME, "MI SCRNI O", " ";
  RSLVSP          MI SCRNI O, RESOLVE, *, *;

  CPYBLA          CTRL-OPCODE, "OPEN";
  CALLX          MI SCRNI O, MI SCRNI O, *;
  CPYBLA          LEFT-RIGHT, "L";
  CPYNV           START-NBR, 0;

SHOW-THE-SCREEN:
  CPYBREP         CTRL-CURSOR-POSITION, "O";
  CPYBLA          CTRL-OPCODE, "WRITE";
  CALLX          MI SCRNI O, MI SCRNI O, *;

  CPYBREP         S-MESSAGE, " ";
  CPYBLA          A-MESSAGE, "B"; /* blank */

GET-THE-SCREEN:
  CPYBLA          CTRL-OPCODE, "READ";
  CALLX          MI SCRNI O, MI SCRNI O, *;

  CMPNV(B)        CTRL-CMD-KEY, 0/EQ(SEARCH);
  CMPNV(B)        CTRL-CMD-KEY, 3/EQ(DONE);
  CMPNV(B)        CTRL-CMD-KEY, 11/EQ(TOGGLE);
  CMPNV(B)        CTRL-CMD-KEY, 12/EQ(DONE);
  CMPNV(B)        CTRL-CMD-KEY, 17/EQ(TOP);
  CMPNV(B)        CTRL-CMD-KEY, 18/EQ(BOTTOM);
  CMPNV(B)        CTRL-CMD-KEY, 27/EQ(PAGE-UP);
  CMPNV(B)        CTRL-CMD-KEY, 28/EQ(PAGE-DOWN);

FUNCTION-KEY-NOT-USED:
  CPYBLAP         S-MESSAGE, "Function key not used", " ";

SHOW-ERROR-MESSAGE:
  CPYBLA          A-MESSAGE, "T"; /* title */
  B              SHOW-THE-SCREEN;

PAGE-UP:
  SUBN(B)         ENTRY-NBR, START-NBR, SCREEN-LINES/NEG(BUILD-SCREEN);
TOP:
  CPYNV(B)        ENTRY-NBR, 1/POS(BUILD-SCREEN);

BOTTOM:
  SUBN            ENTRY-NBR, OPT-RETURN-COUNT, SCREEN-LINES;
  ADDN(SB)        ENTRY-NBR, 1/POS(BUILD-SCREEN), NPOS(TOP);

PAGE-DOWN:
  CMPBLA(B)       S-MORE, "M"/EQ(BUILD-SCREEN);
  CPYNV(B)        ENTRY-NBR, START-NBR/POS(BUILD-SCREEN);

TOGGLE:
  XOR(S)          LEFT-RIGHT, X' 0A' ; /* L <-> R */
  CPYNV(B)        ENTRY-NBR, START-NBR/POS(BUILD-SCREEN);

```

```
ZER(SHOW-THE-SCREEN);
```

```
DONE:
  CPYBLA      CTRL-OPCODE, "CLOSE";
  CALLX      .MI SCRNI O, MI SCRNI O, *;
  RTX        *;
```

Note how the scrolling mechanism invoked by Page Down/Page Up and Top/Bottom function keys are elegantly coded with only six instructions total, belying the notion that scrollable “subfile” handling is hard.

I couldn’t resist a bit of trick coding: If a byte contains “L”, how to flip it to “R”? and if it contains an “R”, how to flip it to an “L”? why, by XORing the byte with x‘0A’, the result of XORing “L” and “R”. Once you’ve caught on, the trick disappears and becomes a valued idiom.

## Building a Screen

We then search for a word and build a screen with as many matches as will fit:

```
SEARCH:
  CMPBLAP(B)  S-WORD, " ", " "/EQ(ENTER-A-WORD);
  CALLI      SEARCH-FOR-WORD, *, .SEARCH-FOR-WORD;
  CPYNV      ENTRY-NBR, 1;

BUI LD-SCREEN:
  CPYNV      START-NBR, ENTRY-NBR;
  CPYNV      LI NE-NBR, 1;
  CPYBLAP    S-MORE, "More. . . ", " ";

SHOW-NEXT-ENTRY:
  CMPNV(B)   LI NE-NBR, SCREEN-LINES /HI (SHOW-THE-SCREEN);
  CMPNV(B)   ENTRY-NBR, OPT-RETURN-COUNT/HI (SHOW-EMPTY-LI NE);
  CPYBLA     S-WORDS (LI NE-NBR), ENTRY-WORD (ENTRY-NBR);
  CPYBLA     S-MEMBERS(LI NE-NBR), ENTRY-MEMBER(ENTRY-NBR);
  CPYBLA     S-LI NES (LI NE-NBR), ENTRY-LI NE (ENTRY-NBR);
```

Here we just fiddle with TEXT to show either the left or the right portion of it:

```
CPYBLA      TEXT, ENTRY-TEXT (ENTRY-NBR);
CMPBLA(B)   LEFT-RI GHT, "L"/NEQ(=+3);
CPYBLAP    S-TEXT, "Text-->", " ";
CPYBLA     S-TEXTS (LI NE-NBR), TEXT( 1: 34);:
CMPBLA(B)   LEFT-RI GHT, "R"/NEQ(=+3);
CPYBLAP    S-TEXT, "<---Text", " ";
CPYBLA     S-TEXTS (LI NE-NBR), TEXT(35: 34);:
```

If we are at the end of the list put “Bottom” on the screen:

```
ADDN(S)     ENTRY-NBR, 1;
CMPNV(B)    ENTRY-NBR, OPT-RETURN-COUNT/NHI (+=2);
CPYBLAP    S-MORE, " Bottom", " ";:
ADDN(SB)    LI NE-NBR, 1/POS(SHOW-NEXT-ENTRY);

SHOW-EMPTY-LI NE:
  CMPNV(B)   LI NE-NBR, SCREEN-LINES/HI (SHOW-THE-SCREEN);
  CPYBREP    S-WORDS (LI NE-NBR), " ";
  CPYBREP    S-MEMBERS(LI NE-NBR), " ";
  CPYBREP    S-LI NES (LI NE-NBR), " ";
  CPYBREP    S-TEXTS (LI NE-NBR), " ";
  ADDN(SB)   LI NE-NBR, 1/POS(SHOW-EMPTY-LI NE);
```

If the index was not found or the word entered was blank, complain accordingly:

```
DCL EXCM * EXCI D(H' 2200' ) BP(INDEX-NOT-FOUND) CV(X' 00000000' ) IMD;
INDEX-NOT-FOUND:
  CPYBLAP    S-MESSAGE, "Index not found. . . ", " ";
  CPYBLA     A-INDEX, "e"; /* error */
  CPYBLA     A-LI BRARY, "e";
  B          SHOW-ERROR-MESSAGE;

ENTER-A-WORD:
  CPYBLAP    S-MESSAGE, "Enter a word to search for. . . ", " ";
  CPYBLA     A-WORD, "G"; /* error */
  B          SHOW-ERROR-MESSAGE;
```

Remember from Chapter 17 that some attributes, e.g. “e” and “G”, automatically change after the screen has been shown (to remove the highlighting).

## Searching for the Word

Up to this point it has all been about formatting, building, and scrolling a screenful of data. All part of the necessary housekeeping, but terribly unexciting (as far as index manipulation goes). Now for interesting part:

```
DCL SYSPTR . THE-LI BRARY;
DCL SYSPTR . THE-I NDEX;

DCL INSPTR . SEARCH-FOR-WORD;
ENTRY SEARCH-FOR-WORD INT;
      CMPBLAP(B) S-LI BRARY, " ", " " /EQ(USE-LI BRARY-LI ST);
      CMPBLAP(B) S-LI BRARY, "*LI BL", " " /EQ(USE-LI BRARY-LI ST);

USE-GI VEN-LI BRARY:
  CPYBLA RESOLVE-TYPE, X' 0401' ;
  CPYBLAP RESOLVE-NAME, S-LI BRARY, " ";
  RSLVSP . THE-LI BRARY, RESOLVE, *, *;

  CPYBLA RESOLVE-TYPE, X' 0EOA' ;
  CPYBLAP RESOLVE-NAME, S-I NDEX, " ";
  RSLVSP . THE-I NDEX, RESOLVE, . THE-LI BRARY, *;
  B FIND-WORDS;

USE-LI BRARY-LI ST:
  CPYBLA RESOLVE-TYPE, X' 0EOA' ;
  CPYBLAP RESOLVE-NAME, S-I NDEX, " ";
  RSLVSP . THE-I NDEX, RESOLVE, *, *;
```

The above code just resolves to the index. No fun yet. Now, the “Find Index Entry” instruction, FNDI NXEN, takes four operands. The first operand is a pointer to receiver area where the instruction will store the entries retrieved. We’ll make room for 4000 entries:

```
DCL SPCPTR . ENTRIES INIT(ENTRIES);
DCL DD ENTRIES (4000) CHAR(120) BDRY(16);
DCL DD ENTRY-WORD (4000) CHAR(25) DEF(ENTRIES) POS( 1) AEO(120);
DCL DD ENTRY-MEMBER(4000) CHAR(10) DEF(ENTRIES) POS(26) AEO(120);
DCL DD ENTRY-LINE (4000) CHAR(05) DEF(ENTRIES) POS(36) AEO(120);
DCL DD ENTRY-TEXT (4000) CHAR(80) DEF(ENTRIES) POS(41) AEO(120);
```

The second operand is the system pointer to the index that we’ve just resolved. The third operand is a pointer to an “option list”:

```
DCL SPCPTR . OPTIONS INIT(OPTIONS);
DCL DD OPTIONS CHAR(16010);
DCL DD OPT-RULE CHAR(2) DEF(OPTIONS) POS( 1);
DCL DD OPT-ARG-LENGTH BIN(2) DEF(OPTIONS) POS( 3) INIT(26);
DCL DD OPT-ARG-OFFSET BIN(2) DEF(OPTIONS) POS( 5) INIT(26);
DCL DD OPT-MAX-COUNT BIN(2) DEF(OPTIONS) POS( 7) INIT(4000);
DCL DD OPT-RETURN-COUNT BIN(2) DEF(OPTIONS) POS( 9);
DCL DD OPT-ENTRY(4000) CHAR(4) DEF(OPTIONS) POS(11);
```

The RULE is one of

- x'0001' = Find equal occurrences of operand 4
- x'0002' > Find occurrences greater than operand 4
- x'0003' < Find occurrences less than operand 4
- x'0004' >= Find occurrences greater than or equal to operand 4
- x'0005' <= Find occurrences less than or equal to operand 4
- x'0006' first Find first occurrence(s)
- x'0007' last Find last occurrence(s)
- x'0008' between Find all entries between the two arguments specified by operand 4

Since the key consists of a 25-character word followed by 15 characters of other information and since we are only searching on the word, we’ll construct arguments having 26 characters each, where the 26<sup>th</sup> character are x'00' and x'FF' for the two arguments respectively. This will make all of the additional

information fall within the range. The ARG-LENGTH should then be 26. The ARG-OFFSET is the offset to the second argument, thus is also 26. MAX-COUNT is the maximum number of entries to retrieve, and RETURN-COUNT will be the number of entries actually satisfying the option rule. If these two numbers are equal there may be more than MAX-COUNT qualifying entries.

A pointer to the arguments is the fourth operand:

```
DCL SPCPTR . ARGUMENTS INIT(ARGUMENTS);
DCL DD ARGUMENTS CHAR(52);
DCL DD ARG-FROM-WORD CHAR(26) DEF(ARGUMENTS) POS( 1);
DCL DD ARG-UPTO-WORD CHAR(26) DEF(ARGUMENTS) POS(27);
```

To cater to a generic search (like for “word\*”) we replace all characters in the arguments from (and including) the asterisk to the end of the word by x‘00’ and x‘FF’ respectively:

```
FIND-WORDS:
CPYBREP ARG-FROM-WORD, X'00';
CPYBREP ARG-UPTO-WORD, X'FF';

TRIML SIZE, S-WORD, " ";
CMPBLA(B) S-WORD(SIZE:1), "*" /NEQ(=+2);
SUBN(SB) SIZE, 1 /POS(=+2), ZER(=+4);:
CPYNV SIZE, 25;:
CPYBLA ARG-FROM-WORD, S-WORD(1:SIZE);
CPYBLA ARG-UPTO-WORD, S-WORD(1:SIZE);:
```

Finally, we specify RULE “between” and go:

```
CPYBLA OPT-RULE, X'0008'; /* BETWEEN */
FNDINXEN .ENTRIES, .THE-INDEX, .OPTIONS, .ARGUMENTS;
B .SEARCH-FOR-WORD;
```

Blank

## Chapter 26

### The AS/400 Memory Explorer

#### Examining Memory without SST

In the present book I have often shown the contents of memory for objects or control blocks. You may wish to be able to do similar things. Now, DST or SST can do that, but in a “clunky” way; and you may not always have access to SST when you need it. In this chapter we shall develop a simple SST “look-a-like” that allows effortless browsing of arbitrary AS/400 memory. We shall build on the “counterfeit” pointer technique of Chapter 7 and on the screen handler of Chapter 18. Here is a typical screenshot of what the result will look like:

```
AS/400 Explorer

Object to explore . . . . .
Context . . . . .
Address CB553B28A3 000000

000000 00020008 00810001 FFCA4AF1 B5000000 .....a...-c1$...
000010 40010000 00000000 CB553B28 A3000020 .....ôí..t...
000020 80000000 00000000 10ADBDD9 B6001000 0.....Y`R]... Space Ptr
000030 00008000 00000000 3E011AC7 7800193F ..0.....Gİ... Space
000040 00008000 00000000 FD11D572 030019FF ..0.....Û·NĖ... Space
000050 00008000 00000000 1022EA47 BA0010FF ..0.....²ā[... Dev Descr
000060 00008000 00000000 2E034719 490004FF ..0.....â·ñ... Library
000070 80000000 00000000 2188453C DC000100 0.....há·ü... Space Ptr
000080 80000000 00000000 DEDA4228 D7000100 0.....ú'â·P... Space Ptr
000090 00000000 00000000 0712C4EB 76000800 .....DÔi... User Prf
0000A0 00008000 00000000 F3BB83E8 E4000AFF ..0.....3]cYU... Queue
0000B0 00000000 00000000 00000000 00000000 .....
0000C0 00000000 00000000 00000000 00000000 .....
0000D0 00000000 00000000 00000000 00000000 .....
0000E0 80000000 00000000 C1FEE99F 84000100 0.....AÜZªd... Space Ptr
0000F0 00000000 00000000 00000000 00000000 .....

F3=Exit F2=Disasm F4=Look F9=Time F10=PCS F12=Goback F17=Top

More...

Mâ a 03/033
```

You can specify the name, type, and context of an object, or give an address. When you press the Enter key, the *AS/400 Explorer* will show the contents of the first 256 bytes of that memory area. Press Page Down/Up to see more of the contents. The data is presented in a hexadecimal form in two side-by-side panels. A third panel shows the contents as straight text (non-displayable characters are shown as •). If a 16-byte block contains a valid pointer, the block is highlighted and the type of the pointer is indicated in a fourth panel. To “follow” a pointer or to see what is at an address, place the cursor over it and press F4; this you can do to any depth. Press F12 (repeatedly - if you wish) to retrace your steps through the previous displays. Timestamps are 64 bits; place the cursor over a timestamp and press F9 to see what time that represents. If you get lost, press F10 to get back to your process’ Process Control Space (which is also where the display starts from if you don’t specify an object or an address).

#### Accessing Arbitrary Memory

We need a “helper” program, **MI ACCESS**, to copy memory from one address to another one. The parameter list is simple:

```
DCL SPCPTR . PARM1 PARM;
DCL DD CONTROL CHAR(21) BAS(. PARM1);
DCL DD TO-ADDRESS CHAR(8) DEF(CONTROL) POS( 1);
DCL DD FROM-ADDRESS CHAR(8) DEF(CONTROL) POS( 9);
DCL DD NBR-OF-BYTES BIN(4) DEF(CONTROL) POS(17);
DCL DD FEEDBACK CHAR(1) DEF(CONTROL) POS(21);

DCL OL PARMS (. PARM1) EXT PARM MIN(1);
```



Specify the 64-bit FROM-ADDRESS, the 64-bit TO-ADDRESS, and the number of bytes to move (in multiples of 16). A feedback byte will contain a space if there was no error or an “E” if the memory page addressed does not exist as a valid virtual address. We use **MI MAKPTR** to turn addresses into valid pointers. A more sophisticated approach would patch MI ACCESS to not needing MI MAKPTR, but let’s keep it simple here and build on what we already have:

```
DCL SPCPTR .ARG1 INIT(POINTER);
DCL DD POINTER CHAR(16) BDRY(16);
DCL SPCPTR .POINTER DEF(POINTER) POS(1);
DCL DD PTR-TYPE CHAR(8) DEF(POINTER) POS(1);
DCL DD PTR-ADDRESS CHAR(8) DEF(POINTER) POS(9);

DCL SPCPTR .FROM;
DCL DD FROM CHAR(256) BAS(.FROM);

DCL SPCPTR .TO;
DCL DD TO CHAR(256) BAS(.TO);

DCL OL MI MAKPTR (.ARG1) ARG;
DCL SYSPTR .MI MAKPTR;

DCL DD RESOLVE CHAR(34);
DCL DD RESOLVE-TYPE CHAR( 2) DEF(RESOLVE) POS( 1) INIT(X' 0000' );
DCL DD RESOLVE-NAME CHAR(30) DEF(RESOLVE) POS( 3);
DCL DD RESOLVE-AUTH CHAR( 2) DEF(RESOLVE) POS(33) INIT(X' 0000' );

ENTRY * (PARMS) EXT;
  CPYBLA FEEDBACK, " ";
  CMPBLA(B) RESOLVE-TYPE, X' 0000' /NEQ(COPY-DATA);
```

If we have not yet resolved the system pointer to MI MAKPTR, we do so now:

```
RESOLVE-TO-PGM:
  CPYBLA RESOLVE-TYPE, X' 0201' ;
  CPYBLAP RESOLVE-NAME, "MI MAKPTR", " ";
  RSLVSP .MI MAKPTR, RESOLVE, *, *;
```

Then we convert the two addresses to two space pointers:

```
COPY-DATA:
  CPYBLA PTR-ADDRESS, FROM-ADDRESS;
  CALLX .MI MAKPTR, MI MAKPTR, *;
  CPYBWP .FROM, .POINTER;

  CPYBLA PTR-ADDRESS, TO-ADDRESS;
  CALLX .MI MAKPTR, MI MAKPTR, *;
  CPYBWP .TO, .POINTER;
```

and copy the data (with tags bits intact):

```
  CPYBWP TO(1:NBR-OF-BYTES), FROM(1:NBR-OF-BYTES);

RETURN:
  RTX *;
```

If we get an error, return an “E” as the feedback byte:

```
DCL EXCM * EXCID(H' 0000' ) BP(CANNOT-ACCESS) CV(X' 00000000' ) IMD;
CANNOT-ACCESS:
  CPYBLA FEEDBACK, "E";
  B RETURN;
```

## Preserving Tag Bits

To be able to inspect system domain memory, we make MI ACCESS a system-state program. MI MAKPTR should then also have the system-state attribute (or at least the inherit-state attribute). Note that we copied the data with the CPYBWP MI-instruction. This instruction copies data that are on 16-byte boundaries with the LQ, STQ RISC-instruction pair preserving the pointer tag bits. When we call MI ACCESS we must be careful to supply a ‘to-address’ with the BDRY(16) attribute. The reason for all this concern about tag bits is that we want to highlight pointers on the display and the tag bits on identify the address as a pointer.

## Screen Layout

The **MI SCRNI O** screen handler we developed in Chapter 18 takes a *screen description* as a parameter. The screen description is a sequence of fields, each described by a 10-byte *field introducer* followed by the field proper. The introducer contains the logical screen attribute, the starting row, column, and size of the field. We'll only show a few fields here and refer to the program for the rest. Here's the screen title:

```
DCL SPCPTR . SCREEN INIT(SCREEN);
DCL DD SCREEN CHAR(10) INIT("T010320015");
DCL DD S-TITLE CHAR(15) INIT("AS/400 Explorer");
DCL DD * CHAR(10) INIT("L010480000");
```

The address line contains two fields, the segment and the offset (and a hidden field to 'rest' the cursor on):

```
DCL DD * CHAR(10) INIT("L050010007");
DCL DD * CHAR(07) INIT("Address . . . . .");
DCL DD A-SEGMENT CHAR(10) INIT("q050090010");
DCL DD S-SEGMENT CHAR(10) INIT(" "); /* segment part of address */
DCL DD A-OFFSET CHAR(10) INIT("q050200006");
DCL DD S-OFFSET CHAR(06) INIT(" "); /* offset part of address */
DCL DD * CHAR(10) INIT("S050270002");
DCL DD * CHAR(02) INIT(" "); /* cursor resting place */
```

## Arrays of Fields

The panel columns are described by “anchor fields” (names ending with “@”):

```
DCL DD * CHAR(10) INIT("L060010006"); DCL DD S-WHERE@ CHAR(06);
DCL DD @1 CHAR(10) INIT("j 060090008"); DCL DD S-WORD1@ CHAR(08);
DCL DD @2 CHAR(10) INIT("j 060180008"); DCL DD S-WORD2@ CHAR(08);
DCL DD @3 CHAR(10) INIT("j 060280008"); DCL DD S-WORD3@ CHAR(08);
DCL DD @4 CHAR(10) INIT("j 060370008"); DCL DD S-WORD4@ CHAR(08);
DCL DD * CHAR(10) INIT("U060480016"); DCL DD S-TEXT@ CHAR(16);
DCL DD * CHAR(10) INIT("L060650015"); DCL DD S-DESCR@ CHAR(15);
DCL DD * CHAR(10) INIT("L060800000"); /* SIZE 69 */

DCL DD * CHAR(10) INIT("L070010006"); DCL DD * CHAR(06);
DCL DD * CHAR(10) INIT("j 070090008"); DCL DD * CHAR(08);
DCL DD * CHAR(10) INIT("j 070180008"); DCL DD * CHAR(08);
DCL DD * CHAR(10) INIT("j 070280008"); DCL DD * CHAR(08);
DCL DD * CHAR(10) INIT("j 070370008"); DCL DD * CHAR(08);
DCL DD * CHAR(10) INIT("U070480016"); DCL DD * CHAR(16);
DCL DD * CHAR(10) INIT("L070650015"); DCL DD * CHAR(15);
DCL DD * CHAR(10) INIT("L070800000");
```

... (repeat for 14 more rows)

Note that the anchor introducers are also named (@1, etc). This structure allows us to define arrays (with 16 elements) over the attributes and values of the fields. The array element offsets (AE0s) are the total size (8 introducers of 10 bytes plus 69 bytes worth of field values) of the description for each row:

```
DCL DD A-WORD1 (16) CHAR(01) DEF(@1 ) POS(1) AE0(149); /* attributes */
DCL DD A-WORD2 (16) CHAR(01) DEF(@2 ) POS(1) AE0(149);
DCL DD A-WORD3 (16) CHAR(01) DEF(@3 ) POS(1) AE0(149);
DCL DD A-WORD4 (16) CHAR(01) DEF(@4 ) POS(1) AE0(149);

DCL DD S-WHERE (16) CHAR(06) DEF(S-WHERE@) POS(1) AE0(149); /* field values */
DCL DD S-WORD1 (16) CHAR(08) DEF(S-WORD1@) POS(1) AE0(149);
DCL DD S-WORD2 (16) CHAR(08) DEF(S-WORD2@) POS(1) AE0(149);
DCL DD S-WORD3 (16) CHAR(08) DEF(S-WORD3@) POS(1) AE0(149);
DCL DD S-WORD4 (16) CHAR(08) DEF(S-WORD4@) POS(1) AE0(149);
DCL DD S-TEXT (16) CHAR(16) DEF(S-TEXT@ ) POS(1) AE0(149);
DCL DD S-DESCR (16) CHAR(15) DEF(S-DESCR@) POS(1) AE0(149);

DCL DD LINE-NBR BIN(2);
DCL DD SCREEN-LINES BIN(2) INIT(16); /* number of array elements */
```

We used a similar structure in Chapter 25. Although superficially similar to a “subfile” it really isn't, because we do not beforehand how much data you want to view. But we do not need to emulate the cumbersome subfile mechanism, just load up each screen panel as we get to it.

The control parameter to **MI SCRNI O** contains the operation code, function key pressed (CTRL-CMD-KEY), and the cursor position (CTRL-CURSOR-ROW and CTRL-CURSOR-COL).

## The Explorer Program (*MI EXPLR*)

We start by resolving system pointers to the sub-programs:

```
ENTRY * EXT;
  CPYBLA      RESOLVE-TYPE, X' 0201' ;
  CPYBLAP     RESOLVE-NAME, "MI ACCESS", " ";
  RSLVSP      .MI ACCESS, RESOLVE, *, *;

  CPYBLAP     RESOLVE-NAME, "MI SCRNI O", " ";
  RSLVSP      .MI SCRNI O, RESOLVE, *, *;

DCL DD RESOLVE CHAR(34);
DCL DD RESOLVE-TYPE CHAR( 2) DEF(RESOLVE) POS( 1);
DCL DD RESOLVE-NAME CHAR(30) DEF(RESOLVE) POS( 3);
DCL DD RESOLVE-AUTH CHAR( 2) DEF(RESOLVE) POS(33) INIT(X' 0000' );
```

## The DATA to Show

The screen will show 256 bytes of data in 16 lines of 16 bytes each. We need an area to hold the current screen's worth of data, organized into 16 lines of either data or pointers:

```
DCL DD DATA CHAR(256) BDRY(16);
DCL DD DATA-LINE(16) CHAR(16) DEF(DATA) POS(1);
DCL PTR .DATA-PTR (16) DEF(DATA) POS(1); /* redefinition */

DCL DD DATA-PTR CHAR(16) BDRY(16);
DCL SPCPTR .DATA DEF(DATA-PTR) POS(1);
DCL DD DATA-ADDRESS CHAR(8) DEF(DATA-PTR) POS(9); /* last 8 bytes of pointer .DATA */
```

Since we are going to be moving to DATA, we need to set the “to-address” for MI ACCESS to be the address of DATA (the number of bytes to move will always be 256):

```
SETSP      .DATA, DATA;
CPYBLA     TO-ADDRESS, DATA-ADDRESS;
CPYNV      NBR-OF-BYTES, 256;

DCL SPCPTR .CONTROL INIT(CONTROL);
DCL DD CONTROL CHAR(21) BDRY(8);
DCL DD TO-ADDRESS CHAR(8) DEF(CONTROL) POS( 1);
DCL DD FROM-ADDRESS CHAR(8) DEF(CONTROL) POS( 9);
DCL DD NBR-OF-BYTES BIN(4) DEF(CONTROL) POS(17);
DCL DD FEEDBACK CHAR(1) DEF(CONTROL) POS(21);

DCL OL      MI ACCESS(. CONTROL) ARG;
DCL SYSPTR .MI ACCESS;
```

## Stacking/Unstacking of Addresses

An important part of making the Explorer user-friendly is the automatic stack/unstack mechanism; whenever you select a new address to explore, the current address is stacked. Pressing F12 unstacks the previous address and you are back to where you started. It is useful to stack/unstack the cursor position too, so that the cursor also returns to where it was:

```
DCL DD STACKED-SEGMENT(500) CHAR(10); /* room for 500 entries */
DCL DD STACKED-OFFSET (500) CHAR( 6);
DCL DD STACKED-CURSOR (500) CHAR( 5);
DCL DD STACK-TOP BIN(2);

CPYNV      STACK-TOP, 0; /* initialize to an empty stack */
SETSP      .PCO, PCO; /* see later for this one */
```

## Showing the Screen

We are now almost ready to show the first screen. First we must *open* the screen:

```
CPYBLA     CTRL-OPCODE, "OPEN";
CALLX      .MI SCRNI O, MI SCRNI O, *;
CPYBREP     CTRL-CURSOR-POSITION, "O";
```

Setting the cursor position to all zeroes has the effect of letting MI SCRNI O decide where to place the cursor (usually at the first input field).

From now on, the flow of control is a loop to write the current screen image, read the screen, and act upon the command keys pressed until F3=Exit is pressed:

```
SHOW-THE-SCREEN:
  CPYBLA      CTRL-OPCODE, "WRITE";
  CALLX      .MI SCRNI 0, MI SCRNI 0, *;

  CPYBREP      S-MESSAGE, " ";
  CPYBLA      A-MESSAGE, "B"; /* blank */
```

After having shown the screen, the message field is cleared, so that the remainder of the code doesn't need to worry about this. Note that data fields on the screens have names starting with "S-" and that named field introducers often have names starting with "A-". The first byte of a field introducer contains the logical attribute. Moving a single character to the field introducer will then only affect the attribute byte.

```
GET-THE-SCREEN:
  CPYBLA      CTRL-OPCODE, "READ";
  CALLX      .MI SCRNI 0, MI SCRNI 0, *;
  CMPNV(B)    CTRL-CMD-KEY, 3/EQ(DONE);
  CMPNV(B)    CTRL-CMD-KEY, 10/EQ(HOME);
```

If either F3 or F10 was pressed we don't care about verifying the screen input, but simply exit the program or go to its "home" position (showing the Process Control Space). Otherwise, it's time to verify that the address has valid syntax (must be a pair of hexadecimal numbers):

```
VERIFY-THE-ADDRESS:
  XLATE      S-SEGMENT, S-SEGMENT, " ", "0";
  VERIFY     SGM-ERR, S-SEGMENT, "0123456789ABCDEF";
  XLATE      S-OFFSET, S-OFFSET, " ", "0";
  VERIFY     OFF-ERR, S-OFFSET, "0123456789ABCDEF";
  ADDN(B)    THE-ERR, SGM-ERR, OFF-ERR/POS(BAD-ADDRESS);

  CVTCH      THE-SEGMENT, S-SEGMENT;
  CPYBLA      S-OFFSET(6:1), "0"; /* set last digit to 0, for 16-byte alignment */
  CVTCH      THE-OFFSET, S-OFFSET;
```

```
DCL DD THE-ADDRESS CHAR(8) BDRY(8);
DCL DD THE-SEGMENT CHAR(5) DEF(THE-ADDRESS) POS(1);
DCL DD THE-OFFSET CHAR(3) DEF(THE-ADDRESS) POS(6);
```

Externally the address parts are represented in hexadecimal "text form" as in "12A6B7...", while internally they are stored in binary form as in X'12A6B7...'. The CVTHC and CVTCH instructions convert back and forth. The names of the instructions are slightly misleading: "CVTHC" converts from binary to text, while "CVTCH" converts from text to binary.

```
DCL DD SGM-ERR BIN(2); /* helper variables for syntax checking */
DCL DD OFF-ERR BIN(2);
DCL DD THE-ERR BIN(2);
```

We finally branch based on the function key pressed:

```
CHECK-FUNCTION-KEY:
  CMPNV(B)    CTRL-CMD-KEY, 0/EQ(ENTER);
  CMPNV(B)    CTRL-CMD-KEY, 2/EQ(DISASSEMBLE);
  CMPNV(B)    CTRL-CMD-KEY, 4/EQ(LOOK);
  CMPNV(B)    CTRL-CMD-KEY, 9/EQ(TIME);
  CMPNV(B)    CTRL-CMD-KEY, 12/EQ(UNSTACK);
  CMPNV(B)    CTRL-CMD-KEY, 17/EQ(TOP);
  CMPNV(B)    CTRL-CMD-KEY, 27/EQ(PAGE-UP);
  CMPNV(B)    CTRL-CMD-KEY, 28/EQ(PAGE-DOWN);

FUNCTION-KEY-NOT-USED:
  CPYBLAP      S-MESSAGE, "Function key not used", " ";
  B            SHOW-ERROR-MESSAGE;
```

Recall that MI SCRNI 0 translates the Enter/ Page up/down keys to command key values 0/27/28. At SHOW-ERROR-MESSAGE, the logical attribute of the message field is set to bright and control returns to the top of the loop:

```
SHOW-ERROR-MESSAGE:
  CPYBLA      A-MESSAGE, "T"; /* title */
  B            SHOW-THE-SCREEN;
```

If the address was not a valid hexadecimal string we complain:

```
BAD-ADDRESS:
  CMPNV(B)      SGM-ERR, 0/EQ(=+2);
  CPYBLA        A-SEGMENT, "g";
  CMPNV(B)      OFF-ERR, 0/EQ(=+2);
  CPYBLA        A-OFFSET, "g";
  CPYBLAP       S-MESSAGE, "Invalid hexadecimal address", " ";
  B             SHOW-ERROR-MESSAGE;
```

Now, we'll flesh out the various actions to invoke.

## Enter: Specifying the Location of the Data

The idea is that if the object field in row three is blank or the cursor is not in part of the screen where you specify the name and context of an AS/400 object, you retrieve data based on the address fields; otherwise we try to locate the object and show its contents:

```
ENTER:
  CMPBLAP(B)    S-SEGMENT, "0", "0"/EQ(HOME);
  CMPBLAP(B)    S-OBJECT, " ", " "/EQ(BUILD-SCREEN);
  CMPNV(B)      CTRL-CURSOR-ROW, 4/HI (BUILD-SCREEN);

RESOLVE-TO-OBJECT:
  CMPBLAP(B)    S-TYPE, " ", " "/NEQ(=+2);
  CPYBLA        S-TYPE, "0201"; /* Default = *PGM */
  XLATE        S-TYPE, S-TYPE, " ", "0";
  VERIFY(B)     THE-ERR, S-TYPE, "0123456789ABCDEF"/POS(BAD-TYPE);
```

The type/subtype must be given in hexadecimal form. We check that the syntax is correct, defaulting to x'0201' \*PGM) if the type field is blank.

```
CVTCH          RESOLVE-TYPE, S-TYPE;
CPYBLA         RESOLVE-NAME, S-OBJECT;
RSLVSP         .THE-OBJECT, RESOLVE, *, *;
```

```
DCL DD THE-OBJECT CHAR(16) BDRY(16);
DCL SYSPTR .THE-OBJECT DEF(THE-OBJECT) POS( 1);
DCL DD OBJ-SEGMENT CHAR(5) DEF(THE-OBJECT) POS( 9); /* redef */
DCL DD OBJ-OFFSET CHAR(3) DEF(THE-OBJECT) POS(14); /* redef */
DCL DD OBJ-1ST-WORD CHAR(4) DEF(THE-OBJECT) POS( 9); /* redef */
DCL DD OBJ-2ND-WORD CHAR(4) DEF(THE-OBJECT) POS(13); /* redef */
```

The object address' segment and offset parts are located in the resolved system pointer. As the offset part contains the object type and authority, we need to clear this information out, as the object really starts at offset zero:

```
SHOW-BASED-ON-SYSTEM-PTR:
  CPYBREP      OBJ-OFFSET, X'00';
SHOW-BASED-ON-ADDRESS:
  CVTHC        S-SEGMENT, OBJ-SEGMENT;
  CVTHC        S-OFFSET, OBJ-OFFSET;
```

BUILD-SCREEN:

We have now a valid address. We'll see a bit later how to retrieve the data at that address and build the screen display. For now, it suffices that we have a common place to branch to (BUILD-SCREEN) where this action is implemented.

If we could not resolve to the object, we issue a suitable error message:

```
DCL EXCM * EXCID(H' 2200' ) BP(OBJECT-NOT-FOUND) CV(X' 00000000' ) IMD;
OBJECT-NOT-FOUND:
  CPYBLA        A-OBJECT, "e";
  CPYBLA        A-TYPE, "e";
  CPYBLAP       S-MESSAGE, "Object not found...", " ";
```

SHOW-ERROR-MESSAGE:

...

Additionally, if the object type was not a valid hexadecimal string, we complain:

```
BAD-TYPE:
  CPYBLA        A-TYPE, "e";
  CPYBLAP       S-MESSAGE, "Invalid object type", " ";
  B             SHOW-ERROR-MESSAGE;
```

## Paging Down/Up to See More

When the address fields were validated, the current segment and offset corresponding to that address were placed in THE-SEGMENT and THE-OFFSET. To page up (*i.e.* move to a lower address) all we need to do is to subtract 256 (x'100') from the current offset, making sure that the offset does not go below zero:

```
PAGE-UP:
  CMPBLA(B)      THE-OFFSET, X' 000100' /HI (=+2);
TOP:
  CPYBLA         THE-OFFSET, X' 000100' ;:
  SUBLC(S)       THE-OFFSET, X' 000100' ;:
```

Going to the “top” of the data for the segment, to handle F17=Top, simply means forcing the offset to zero.

```
SET-OFFSET:
  CVTHC          S-OFFSET, THE-OFFSET;
  B              BUI LD-SCREEN;
```

To page down (*i.e.* move to a higher address) all we need to do is to add 256 (x'100') to the current offset, making sure that the offset does not flow out of the segment (we simply wrap around to the beginning):

```
PAGE-DOWN:
  CMPBLA(B)      THE-OFFSET, X' FFFF00' /NHI (=+2);
  CPYBLA         THE-OFFSET, X' FFFF00' ;:
  ADDLC(S)       THE-OFFSET, X' 000100' ;:
  B              SET-OFFSET;
```

## The HOME or PCS Position

The Process Control Space (the PCS) and its embedded associated space (the Process Communications Object, the PCO) are home to many interesting pointers and address and are good starting points for further exploration. To get the address of the PCS, we simply base some data on the PCO using the **BASPCO** attribute:

```
DCL DD PCO CHAR(256) BASPCO;
DCL DD PCO-POINTER CHAR(16) BDRY(16);
  DCL SPCPTR .PCO          DEF(PCO-POINTER) POS(1);
  DCL DD PCO-ADDRESS CHAR(8) DEF(PCO-POINTER) POS(9);
```

then set a space pointer to that data:

```
SETSPP      .PCO, PCO;
```

This we did during initialization of the program. To implement the “home” function (F10=PCS) we clear the stacked addresses and cursor positions, set the address fields to address part of the PCO-pointer and clear the offset par, then finally go and build the screen as usual:

```
HOME:
  CPYNV        STACK-TOP, 0;
  CPYBLA       THE-ADDRESS, PCO-ADDRESS;
  CPYBREP      THE-OFFSET, X' 00' ;

SET-PCS-ADDR:
  CVTHC        S-SEGMENT, THE-SEGMENT;
  CVTHC        S-OFFSET, THE-OFFSET ;
  B            BUI LD-SCREEN;
```

## Look at Memory at Another Address

Placing the cursor on an address (or a highlighted pointer) and pressing F4=Look should have the following functionality: 1) check that the cursor is within the data area, 2) stack the current address, and 3) go to the address selected and display the data there. First check if the cursor position is where it should be:

```
DCL INSPTR .CHECK-WITHIN-AREA;
ENTRY      CHECK-WITHIN-AREA INT;
  SUBN(B)   LI NE-NBR, CTRL-CURSOR-ROW, 5/NPOS(OUTSIDE-AREA);
  CMPNV(B)  LI NE-NBR, SCREEN-LINES /HI (OUTSIDE-AREA);

  CMPNV(B)  CTRL-CURSOR-COL, 10/LO(OUTSIDE-AREA);
  CMPNV(B)  CTRL-CURSOR-COL, 45/HI (OUTSIDE-AREA);
```

```

        CPYBLA      DATA-WORDS, DATA-LINE(LINE-NBR);
        B           .CHECK-WITHIN-AREA;

OUTSIDE-AREA:
        CPYBLAP     S-MESSAGE, "Place cursor first.", " ";
        B           SHOW-ERROR-MESSAGE;

```

As a side effect, the data line is moved to a holding area with easy access to the four 32-bit words making up the 16-byte data chunk.

```

DCL DD DATA-WORDS CHAR(16) BDRY(16);
DCL DD DATA-WORD(4) CHAR(4) DEF(DATA-WORDS) POS(1);

```

The F4=Look implementation then goes like this:

```

LOOK:
        CALLI       CHECK-WITHIN-AREA, *, .CHECK-WITHIN-AREA;

```

Now stack the current address and cursor position:

```

        CMPNV(B)     STACK-TOP, 500/NLO(=+2);
        ADDN(S)      STACK-TOP, 1;
        CPYBLA       STACKED-SEGMENT(STACK-TOP), S-SEGMENT;
        CPYBLA       STACKED-OFFSET (STACK-TOP), S-OFFSET;
        CPYBLA       STACKED-CURSOR (STACK-TOP), CTRL-CURSOR-POSITION;

```

The address selected with the cursor could be in either the left-hand or the right-hand data panel (or the data line could contain a pointer - extending over both sides of the panel). Use the value of cursor column to decide which one, and pick up the two data words that make up an address:

```

        CMPPTRT(B)   .DATA-PTR(LINE-NBR), */NEQ(LOOK-VIA-POINTER);
        CMPNV(B)     CTRL-CURSOR-COL, 27/HI (LOOK-RIGHT-HAND-SIDE);

LOOK-LEFT-HAND-SIDE:
        CPYBLA       OBJ-1ST-WORD, DATA-WORD(1);
        CPYBLA       OBJ-2ND-WORD, DATA-WORD(2);
        B           SHOW-BASED-ON-ADDRESS;

LOOK-RIGHT-HAND-SIDE:
LOOK-VIA-POINTER:
        CPYBLA       OBJ-1ST-WORD, DATA-WORD(3);
        CPYBLA       OBJ-2ND-WORD, DATA-WORD(4);
        CMPPTRT(B)   .DATA-PTR(LINE-NBR), SYSPTR/
                        EQ(SHOW-BASED-ON-SYSTEM-PTR),
                        NEQ(SHOW-BASED-ON-ADDRESS);

```

## Detecting a Pointer

When we declared the DATA variable, we set it up to be considered both as an array of 16-byte ‘data lines’ and as an array of pointers:

```

DCL DD DATA CHAR(256) BDRY(16);
DCL DD DATA-LINE(16) CHAR(16) DEF(DATA) POS(1);
DCL PTR .DATA-PTR (16) DEF(DATA) POS(1); /* redefinition */

```

The copy function of MIACCESS preserved the tag bits associated with every pointer. There is an MI-instruction, “Compare Pointer Type” (**CMPPTRT**), that can be used to test if its first operand contains a valid pointer and, if so, which type of pointer.

### The **CMPPTRT** MI-instruction

The instruction compares the pointer type currently in operand 1 with the type identified by the 1-byte value in operand 2:

```

        CMPPTRT(B)   Pointer, Type/EQ(Match);

```

Operand 2 can be NULL (“\*”). In that case the EQ (equal) condition means that a valid pointer does not exist in operand 1. If operand 2 is not NULL, the EQ/NEQ (equal/not equal) condition means that a pointer of the requested type does/doesn’t exist in operand 1. Possible type designators are:

```

x'00'   A pointer does not exist at this location
x'01'   System pointer

```

x'02'	Space pointer
x'03'	Data pointer
x'04'	Instruction pointer
x'05'	Invocation pointer
x'06'	Procedure pointer
x'07'	Label pointer
x'08'	Suspend pointer
x'09'	Synchronization pointer
x'0A'	SOM Object pointer
x'0B'	SOM Behavior pointer
x'0C'	Method pointer

So, to test if the location contains a pointer, we coded:

```
CMPPTR(B) . DATA-PTR(LINE-NBR), */NEQ(LOOK-VIA-POINTER); /* branch if not NULL */
```

To test if the location contains a system pointer, we coded:

```
CMPPTR(B) . DATA-PTR(LINE-NBR), SYSPTR/EQ(SHOW-BASED-ON-SYSTEM-PTR),
NEQ(SHOW-BASED-ON-ADDRESS);

DCL DD SYSPTR CHAR(1) INIT(X'01');
DCL DD SPCPTR CHAR(1) INIT(X'02');
```

## Unstacking to Previous Address

If the stack is empty we just go “home” and show the PCS, otherwise we restore the address and cursor position from the stack and remove that stack entry, then go build the screen image:

```
UNSTACK:
CMPNV(B) STACK-TOP, 1/LO(HOME);
CPYBLA S-SEGMENT, STACKED-SEGMENT(STACK-TOP);
CPYBLA S-OFFSET, STACKED-OFFSET(STACK-TOP);
CPYBLA CTRL-CURSOR-POSITION, STACKED-CURSOR(STACK-TOP);
SUBN(SB) STACK-TOP, 1/NNAN(BUILD-SCREEN);
```

## Displaying Timestamps

If you place the cursor on what looks like a valid timestamp, we convert the 64-bit timestamp into a date and time value as explained back in Chapter 4:

```
TIME:
CALLI CHECK-WITHIN-AREA, *, .CHECK-WITHIN-AREA;
CMPNV(B) CTRL-CURSOR-COL, 27/HI(TIME-RIGHT-HAND-SIDE);

TIME-LEFT-HAND-SIDE:
CPYBLA THE-TIME-HI, DATA-WORD(1);
CPYBLA THE-TIME-LO, DATA-WORD(2);
B CONVERT-AND-SHOW-TIME;

TIME-RIGHT-HAND-SIDE:
CPYBLA THE-TIME-HI, DATA-WORD(3);
CPYBLA THE-TIME-LO, DATA-WORD(4);

CONVERT-AND-SHOW-TIME: /* same code as in Chapter 4 */
...
DCL DD THE-TIME CHAR(8);
DCL DD THE-TIME-HI BIN(4) UNSGND DEF(THE-TIME) POS(1);
DCL DD THE-TIME-LO BIN(4) UNSGND DEF(THE-TIME) POS(5);

NOT-A-TIME:
CPYBLAP S-MESSAGE, "Not a time...", " ";
B SHOW-ERROR-MESSAGE;
```

## Disassembling Instructions

Leaving the door open for a future enhancement:

```
DISASSEMBLE:
CPYBLAP S-MESSAGE, "Coming soon...", " ";
B SHOW-ERROR-MESSAGE;
```



## Exiting the Program

Close the screen and return. Note that we do not deactivate the program, so if you call it again later in the same job you get right back to where you were:

```
DONE:
  CPYBLA      CTRL-OPCODE, "CLOSE";
  CALLX      .MI SCRNI 0, MI SCRNI 0, *;
  RTX        *;
```

## Building the Screen Image

First, get the address from the address fields on the screen:

```
BUI LD-SCREEN:
  CPYBLAP     S-MORE, "More...", " ";
  CVTCH      THE-SEGMENT, S-SEGMENT;
  CPYBLA      S-OFFSET(6:1), "0"; /* force bdry(16) */
  CVTCH      THE-OFFSET, S-OFFSET;
```

Then, call MI ACCESS to copy the 256 bytes into DATA:

```
GET-DATA-TO-SHOW:
  CPYBLA      FROM-ADDRESS, THE-ADDRESS;
  CALLX      .MI ACCESS, MI ACCESS, *;
  CMPBLA(B)   FEEDBACK, " " /NEQ(CANNOT-ACCESS);
```

For each of the 16 lines of data, show the four data words as hexadecimal values and as straight text. If the data line contains a pointer, determine the pointer type and set the screen attribute and pointer description text accordingly:

```
      CPYNV      LI NE-NBR, 1;
BUI LD-SCREEN-LI NE:
  CMPNV(B)      LI NE-NBR, SCREEN-LI NES/HI (SHOW-THE-SCREEN);
  CPYBLA        DATA-WORDS, DATA-LI NE(LI NE-NBR);

  CPYBREP       S-DESCR (LI NE-NBR), " ";
  CPYBLA        THE-ATTR, "j";
  CMPPTRT(B)    .DATA-PTR(LI NE-NBR), */EQ(=+2);
  CALLI         GET-POI NTER-DESCR, *, .GET-POI NTER-DESCR;

  XLATEWT       S-TEXT (LI NE-NBR), DATA-WORDS, DI SPLAYABLE-CHARS;

  CPYBLA        A-WORD1 (LI NE-NBR), THE-ATTR;
  CVTHC         S-WORD1 (LI NE-NBR), DATA-WORD(1);
  CPYBLA        A-WORD2 (LI NE-NBR), THE-ATTR;
  CVTHC         S-WORD2 (LI NE-NBR), DATA-WORD(2);
  CPYBLA        A-WORD3 (LI NE-NBR), THE-ATTR;
  CVTHC         S-WORD3 (LI NE-NBR), DATA-WORD(3);
  CPYBLA        A-WORD4 (LI NE-NBR), THE-ATTR;
  CVTHC         S-WORD4 (LI NE-NBR), DATA-WORD(4);

  CVTHC         S-WHERE (LI NE-NBR), THE-OFFSET;
  ADDLC(S)      THE-OFFSET, X' 000010' ;
  ADDN(SB)      LI NE-NBR, 1/POS(BUI LD-SCREEN-LI NE);
```

To ensure that non-displayable characters do not wreak havoc with the screen attributes, we translated the original data to displayable text through this table:

```
DCL DD DI SPLAYABLE-CHARS CHAR(256);
DCL DD * (16) CHAR(16) DEF(DI SPLAYABLE-CHARS) POS(1) INI T
      (X' B3B3B3B3B3B3B3B3B3B3B3B3B3B3B3B3', /* B3 is a fat dot */
      X' B3B3B3B3B3B3B3B3B3B3B3B3B3B3B3B3',
      ""
      X' E0B3E2E3E4E5E6E7E8E9EAEBCEDDEEFF',
      X' F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEB3' );
```

If MI ACCESS reported that the data at the virtual memory address given could not be accessed, we complain:

```
CANNOT-ACCESS:
  CPYBLAP     S-MESSAGE, "Cannot access address...", " ";
  B          SHOW-ERROR-MESSAGE;
```

## Determining the Pointer Description

If the pointer is a space pointer we use “Space Ptr” as description. If the pointer is not a space pointer or a system pointer we just call it “Pointer”:

```
DCL INSPTR .GET-POINTER-DESCR;
ENTRY      GET-POINTER-DESCR INT;
  CPYBLA    THE-ATTR, "C"; /* Choice */
  CPYBLAP    S-DESCR (LINE-NBR), "Pointer", " ";

  CMPPTRT(B) .DATA-PTR(LINE-NBR), SPCPTR/NEQ(=+2);
  CPYBLAP    S-DESCR (LINE-NBR), "Space Ptr", " ";

  CMPPTRT(B) .DATA-PTR(LINE-NBR), SYSPTR/NEQ(.GET-POINTER-DESCR);
```

THE-ATTR is the logical attribute byte to use for the data words on the current line:

```
DCL DD THE-ATTR CHAR(1);
```

For system pointers we can do better, as the object type is stored in the 15<sup>th</sup> byte of the pointer. We use a table (TYPE-ENTRY(36) inside TYPE-DESCR) with entries for every object type known. Searching the table for the type extracted from the system pointer yields the proper description to use:

```
  CPYBLA    THE-ATTR, "0"; /* Option */
  CPYBLA    THE-WORD, DATA-WORD(4);
  CVTHC     TYPE-DESCR(456:2), THE-WORD(3:1); /* Type sentinel */
  SCAN      NBR, TYPE-DESCR, TYPE-DESCR(456:2);
  ADDN(S)    NBR, 2;
  CPYBLAP    S-DESCR (LINE-NBR), TYPE-DESCR(NBR:11), " ";
  B         .GET-POINTER-DESCR;

DCL DD TYPE-DESCR CHAR(468); /* 36 times 13 */
DCL DD TYPE-ENTRY(36) CHAR(13) DEF(TYPE-DESCR) POS(1) INIT
  ("01Access Grp", "02Program", "03Module", "04Library",
  "05? Ptr", "06Byte Spc", "07Journal", "08User Prf",
  "09Jrnl Port", "0AQueue", "0BData Space", "0CData Index",
  "0DCursor", "0EIndex", "0FCommit", "10Dev Descr",
  "11Line Descr", "12Ctrl Descr", "13Dump Space", "14Cls of Serv",
  "15Mode", "16NetW Intf.", "17Conn. List", "18Queue Spc",
  "19Space", "1AProc Ctrl", "1BAuth List", "1CDictionary",
  "1DNetW Srv", "1EStream File", "1FDIstr. File", "20SOM Object",
  "21Object Grp", "81Machine Ctx", "85Stream Obj.", "??System Ptr");

DCL DD NBR BIN(2);
DCL DD THE-WORD CHAR(4);
```

The technique used to search the table is to install the type we are looking for as a “sentinel” in an extra entry at the end of the table i.e. where the ‘??’ resides, and then SCAN for the type. What this achieves for us is that if the SCAN fails to find the pointer type within the first 35 entries, it will eventually find it in position 36 and return ‘System Ptr’ as the description by default.

This completes the development of the AS/400 Explorer.

## Ready-made AS/400 Explorer

For those that do not have access to SST, the ‘programs area’ contains a save file, **MI EXPLR. exe**, with the programs needed for the Explorer. This file is a self-extracting PC executable. Execute it on your PC providing the usual password; upload the resulting save file, **MI EXPLR. sav**, to your AS/400; restore the objects from saved library LSVALLGAARD to your library. The owner is QDBSHR. Change this to suit.

Blank

## Chapter 27

### Inside a Save File

#### Why Save-Files?

The object-based architecture of the AS/400 creates a problem: most objects are not “files” in the usual sense and it makes no sense to “read” or “write” them. So how does one backup or save such objects? The answer to this problem is the *save file*. A save file is a “container” that can hold objects of different types in a special common format. From this container, the objects can then later be reconstructed (*restored*). Since a save file is a *file* object, it can be read or written to as an ordinary file. The save file can thus be transferred to an external medium (*e.g.* tape storage) for archival purposes. A save file (as any other file) can also be easily transferred to another system. Since a save file is itself an object, it consists of several sub-objects and segments just like most other objects. This internal format is quite different from the format of the save file when written to the external medium. In this Chapter we shall explore both formats.

#### The Internal Save Format

To understand the workings of save files, we start by saving the arbitrary objects in the library THELIB to a new save file named SAVEFILE. We use this command:

```
Type command, press Enter.
====> SAVOBJ OBJ(*ALL) LIB(THELIB) DEV(*SAVF) SAVF(SAVEFILE) OUTPUT(*PRINT)
```

The printed output tells us that nine programs and a source file were saved:

```
5769SS1 V4R4M0 990521          Save Object - Object Information
Save file . . . . . : SAVEFILE      Save file library . . . : THELIB
Target release . . . : V4R4M0       Storage . . . . . : *KEEP
Save access paths . . : *NO         Save file data . . . . : *YES
Data compressed . . . : No
Library . . . . . : THELIB
Save date/time . . . : 06/01/01    10:19:28

Object      Type      Attribute  Saved      Size  Owner
MIADSUMI    *PGM                YES        28672  THEUSER
MIAES       *PGM                YES        65536  THEUSER
MICBSRT     *PGM                YES        20480  THEUSER
MICPGM1     *PGM                YES        24576  THEUSER
MICPGM2     *PGM                YES        24576  THEUSER
MIDCEXC     *PGM                YES        32768  THEUSER
MIFAES      *PGM                YES        32768  THEUSER
MIFSORT     *PGM                YES        32768  THEUSER
MIGENUUID   *PGM                YES        16384  THEUSER
QMISRC      *FILE    PF        YES        294912 THEUSER
```

The **DMPYSYOBJ** command will tell us the address of the **SAVEFILE** file object. Inspecting memory at that address (you can use SST for this or the **MIEXPLR** from Chapter 26) shows us

```
Address 00A60B3C5D 000000
000000 00010008 00808000 00A60B3C 5D000000  ....00..w..)...
000010 C0010000 00000000 00A60B3C 5D000100 {.....w..)... ← associated space
000020 80001901 E2C1E5C5 C6C9D3C5 40404040 0...SAVEFILE
000030 40404040 40404040 40404040 40404040 0.....
000040 40408000 00000F00 00000008 00400301 0.....
```

The address at offset x'18' points to the primary associated space for the file object.

```
Address 00A60B3C5D 000100
000100 40000000 00000370 00001470 000000B0  ....0...0...^
000110 00000140 000001C6 00000260 00000000  ... ..F...-... ← relative offset
```

The BIN(4) value at offset x'118' is itself a *relative* (with respect to the space starting at absolute offset x'100') offset to a *dump space* descriptor information block:

```
Address 00A60B3C5D 000360 ( x'100' + x'260' )
000360 00000002 00300000 00000000 00003000  ....
000370 00008000 00000000 00A60B3C 5D0019FF  ..0.....w..)...
000380 00000000 00000000 318A4197 C7001300  ....«..pG... ← Dump Space ptr
```

## The Dump Space

The actual data does not reside in the file object, but in a so-called *dump space*. Dump spaces are MI-objects (of type x'13' and subtype x'90') that provide a storage area within the machine for a dump of system objects. Dump spaces are divided into 512-byte logical blocks. When a dump space is placed on an external medium as a file (or is viewed as a file) an additional 16-byte control section is appended to the end of each block, making the external block size 528 bytes.

## Dump Space Management Instructions

A set of blocked MI-instructions is used to manage dump spaces. Because these instructions are mostly blocked we'll only briefly mention them here for completeness. They are:

- Create Dump Space (**CRTDMPS**). Creates a dump space.
- Destroy Dump Space (**DESDMPS**). Removes a dump space.
- Insert Dump Data (**INSDMPD**). Inserts blocks of dump data into a dump space.
- Materialize Dump Space (**MATDMPS**). Materializes the current attributes of a dump space.
- Modify Dump Space (**MODDMPS**). Modifies the attributes of a dump space.
- Retrieve Dump Data (**RETDMPD**). Retrieves a number of 512-byte blocks of dump data from a dump space.

## Checksums

The 16-byte control section following each 512-byte block contains various checksums to protect the integrity of the data dumped. There are two separate checksums, one for each block and one for the dump space as a whole. The first checksum (for each block) provides a simple correctness check for the data when stored on external media. The second checksum (over the entire file) is calculated in a very complicated way (again, someone fell in to the trap believing complication equals security) and is meant to protect against tampering with the data. It is clear that the normal protection that OS/400 provides is not operative when the save file has been transferred to a different system (e.g. a PC or even another AS/400 where you may have more authority). We shall say more about these checksums later on in this chapter.

## Dump Space Header

The internal 30-character name of the dump space is the concatenation of the save file name, the library name, and a sequence number:

```
Address 318A4197C7 000000
000000 00010010 00818000 318A4197 C7000000 .....a0...«.pG...
000010 C0010000 00000000 1918266A 12000020 f.....
000020 80001390 E2C1E5C5 C6C9D3C5 4040E3C8 00..°SAVEFILE TH ← name
000030 C5D3C9C2 40404040 F0F14040 40404040 ELIB 01
000040 40408000 00000FE0 00000520 00403000 0.....\.....
```

At offset x'500' we find the device type:

```
000500 00000000 C4C9E2D2 00000000 00000000 ...DISK... ← device type
```

At offset x'1000' we find the address of the first dump space data segment:

```
001000 2DF9682A 1F000000 00000000 00000000 .9ç.....
```

The fact that this is an address rather than a pointer means that it is not possible to obtain an MI space pointer to the data in the normal manner. We'll have to manufacture a pointer using the technique from chapter 7.

## The Dump Space Data Pages

The data portion of the dump space resides in an unnamed internal object in the system domain. The address of the data object was found in the dump space header as shown above. The first page (4096 bytes) of this data object only contains a link back to the dump space:

```
Address 2DF9682A1F 000000 is in system domain
000000 000D0508 00B10000 318A4197 C7000000 .....f...«.pG... ← back link
```

The second page (at offset x'1000') contains a timestamp (in the usual 64-bit internal format) and the system model number and processor type:

```
001000 828A8EB8 DA3B0000 00000000 30000000 b«p½'..... ← timestamp
001010 F1F5F040 F9F4F0F1 40404040 40404040 150 9401 ← model
```

## The Dump Descriptor

The third page (at offset x'2000') contains the so-called *load-dump object descriptor*. It has the name **QSRDSSPC.1** (Save/Restore Descriptor Space no. 1). The BIN(4) word preceding the name contains all ones:

```
002000 FFFFFFFF D8E2D9C4 E2E2D7C3 4BF14040 ....QSRDSSPC.1
002010 40404040 40404040 40404040 40404040
002020 404019DB 00000000 00000000 00000000 .û.....
002030 00000000 00000000 00000000 00000000 .....
002040 00000000 00000210 0000001F 00000200 .....
002050 03013600 36000000 2F000000 00000000 .....
002060 00200000 00000001 00000000 00000000 .....
002070 00000000 00000000 00000000 00000000 .....
002080 00000000 00000000 00000000 00000000 .....
002090 00000000 0000D361 C440D6C2 D1C5C3E3 .....L/D OBJECT
0020A0 40C4C5E2 C3D9C9D7 E3D6D940 40400000 DESCRIPTOR ..
0020B0 00000000 00000000 0000C4C9 E2D20000 .....DISK..
0020C0 828A8EB8 B36E0000 0000FF00 00000020 b«p½.>..... ← number of units
0020D0 00000001 00000018 00000000 00000000 .....
```

The size of the descriptor is given as a number of *units* (each unit is 512 bytes long):

```
DCL SPCPTR .DESCR
DCL DD DUMP-DESCRIPTOR CHAR(4096) BAS(.DESCR);
DCL DD DESCR-FFFFFFFF CHAR(4) DEF(DUMP-DESCRIPTOR) POS( 1);
DCL DD DESCR-DUMP-ID CHAR(10) DEF(DUMP-DESCRIPTOR) POS( 5);
DCL DD DESCR-UNITS BIN(4) DEF(DUMP-DESCRIPTOR) POS(205);
```

## Dump Descriptor Catalog

The dump descriptor provides a *catalog* showing which objects it contains. The catalog starts on the next page within the descriptor. The catalog is known as a save/restore descriptor space (type x'19DB'):

```
003000 00010018 00C08000 0C30FB73 4B000000 .....{0...ÛË....
003010 00010000 00000000 0C30FB73 4B000100 .....ÛË.... ← offset to header
003020 800019DB D8E2D9C4 E2E2D7C3 4BF14040 0...ÛQSRDSSPC.1
003030 40404040 40404040 40404040 40404040
003040 4040A000 00002F00 00000018 FF1C0301 µ.....
003050 828A8EB5 F7E10000 0DBF8EE4 BD000000 b«p§7...xpu... ← owner: QSYS addr
003060 00000000 00000000 00000000 00000000 .....
003070 0C30FB73 4B000000 00020000 01000000 ..ÛË.....
```

The catalog header contains the name of the library (the context) where the saved objects reside, a timestamp, and the serial number of the system on which the save file was first created. If you move the save file to another system, this original serial number is not changed. The serial number thus tracks the original creator:

```
003100 013FE3C8 C5D3C9C2 40404040 40404040 ..THELIB ← saved library
003110 40404040 40404040 40404040 40404040
003120 0401828A 8EB62656 80000000 000A0000 ..b«p¶.î0..... ← timestamp
003130 00000040 40F1F0F5 F5F5F5D9 00005CE2 ... 105555R...*S ← serial number
003140 E8E2E5C1 D3404040 00000009 D5000000 YSVAL ....N... ← number of entries
```

The header is followed by a number of 151-byte *catalog entries*. The number of entries is given in the catalog header:

```
DCL DD CATALOG CHAR(...) BAS(...);
DCL DD DUMP-TYPE CHAR(2) DEF(CATALOG) POS( 35);
DCL DD CATALOG-HDR CHAR(80) DEF(CATALOG) POS(257);
DCL DD DUMP-CONTEXT CHAR(30) DEF(CATALOG) POS(259);
DCL DD DUMP-CONTEXT-TYPE CHAR(2) DEF(CATALOG) POS(289);
DCL DD DUMP-TIMESTAMP CHAR(8) DEF(CATALOG) POS(291);
DCL DD DUMP-SERIAL-NBR CHAR(7) DEF(CATALOG) POS(310);
DCL DD DUMP-NBR-ENTRIES BIN(4) DEF(CATALOG) POS(329);
```

The context-type (following the name!) is either x'0401' for a regular library or x'04C1' for a QTEMP. Each catalog entry has this format:

```
DCL DD DUMP-ENTRY CHAR(151);
DCL DD DUMP-OBJ-NAME CHAR(30) DEF(DUMP-ENTRY) POS( 1);
DCL DD DUMP-OBJ-TYPE CHAR(2) DEF(DUMP-ENTRY) POS(31);
DCL DD DUMP-OBJ-USER CHAR(30) DEF(DUMP-ENTRY) POS(33);
DCL DD * CHAR(13) DEF(DUMP-ENTRY) POS(63);
DCL DD DUMP-OBJ-MORE BIN(4) DEF(DUMP-ENTRY) POS(72);
DCL DD DUMP-OBJ-INFO BIN(4) DEF(DUMP-ENTRY) POS(76);
DCL DD * CHAR(76) DEF(DUMP-ENTRY) POS(80);
```

e.g.:

```
003150 D4C9C1C4 E2E4D4C9 40404040 40404040 MIADSUMI ← name
003160 40404040 40404040 40404040 40400201 .. ← type (and subtype)
003170 E3C8C5E4 E2C5D940 40404040 40404040 THEUSER ← owning user profile
003180 40404040 40404040 40404040 40400000 ..
003190 00000000 00700000 00000000 000CB300 ..... ← offset to object info
0031A0 00050000 00000000 00000000 00000000 .....
0031B0 00000000 00000000 00000000 00000000 .....
0031C0 00000000 00000000 00000000 00000000 .....
0031D0 00000000 00000000 00000000 00000000 .....
0031E0 00000000 000000 00000000 00000000 .....
```

## Extended Object Information

Additional information about each object is also stored in the catalog. As different types of objects have different types of additional information, the additional information is stored as a list of variable length items. The offset (from the beginning of the catalog header) to the start of the list is given by the **DUMP-MORE-INFO** variable. Each item is preceded by its BIN(4) length. A length of zeroes signals the end of the list. For the example above, we have an offset of x'0CB3' which, when added to x'3100', points to x'3DB3'):

```
003DB0 00 00003240 40404040 40404040 .... size x0032
003DC0 40404040 40404040 40404040 40404040
003DD0 40404040 40404040 40404040 40404040
003DE0 40404040 40404040 40000000 A8404040 ...y size x00A8
003DF0 40404040 40404040 40404040 40404040
003E00 4040F140 40404040 404040F5 F7F6F9E2 1 5769S
003E10 E2F1E5F4 D9F4D4F0 E5F4D9F4 D4F0F1F0 S1V4R4M0V4R4M010
003E20 F1F0F6F0 F1F1F0F1 F1F0F440 40404040 10601101104
003E30 40404040 40404040 40404040 40404040
003E40 40404040 40404040 40404040 40404040
003E50 40404040 40404040 40404040 40404040
003E60 40404040 40000000 00000000 00000000 .....
003E70 00000000 00000000 00000000 00000000 .....
003E80 00000000 00404000 00000000 00000000 .....
003E90 00000000 00000000 1AE3C8C5 E4E2C5D9 .....THEUSER size x001A
003EA0 404040E2 E8E2E3C5 D4404000 00000000 SYSTEM ....
003EB0 00000000 00000000 00000000 00000000 ..... last (12-byte) item
003EC0 000000 000000 00000000 00000000 .....
```

Four pieces of information are stored here: 1) the 50-character (x'32') text description (happens in this case to be all blanks for this object), 2) information about the compiler used to create the program, 3) original *creating* user profile (not *owning* user profile) and name of system where the object was first created, and 4) some vacuous text. This information is not changed when the object is moved to another system. The specific information here is for a *program* object.

For other types of object, a different selection of information is stored. None of this is, of course, documented anywhere. Other offsets are even present for certain types. E.g. for "file" objects the **DUMP-OBJ-MORE** offset points to additional data, including detailed information about each member of the file. As the devil lurks in the details we'll write a simple program to display the object information for an object in a save file. The program, **MIOBJSAV**, illustrates how to find your way around a save file.

## Show Object in a Save File, **MIOBJSAV**

Given a save file (on the library list path) and an object of a given type saved to this file, find the extended object information.

First, the three parameters to **MIOBJS**AV:

```
DCL SPCPTR .P1 PARM; DCL DD SAVE-FILE CHAR(10) BAS(.P1);
DCL SPCPTR .P2 PARM; DCL DD SAVED-OBJ CHAR(10) BAS(.P2);
DCL SPCPTR .P3 PARM; DCL DD SAVED-TYPE CHAR( 4) BAS(.P3);

DCL OL PARMS(.P1, .P2, .P3) PARM EXT MIN(3);

DCL DD OBJECT-TYPE CHAR(2);
DCL DD OBJECT-NAME CHAR(30);

ENTRY * (PARMS) EXT;
  CPYBLAP OBJECT-NAME, SAVED-OBJ, " ";
  CVTCH OBJECT-TYPE, SAVED-TYPE;
```

The object type should be given as a four-byte hexadecimal value, *e.g.* 0201 for \*PGM, hence the conversion with CVTCH. We need to resolve system pointers to the MIMAKPTR helper-program and to the save file:

```
DCL DD RESOLVE CHAR(34);
  DCL DD RESOLVE-TYPE CHAR( 2) DEF(RESOLVE) POS( 1);
  DCL DD RESOLVE-NAME CHAR(30) DEF(RESOLVE) POS( 3);
  DCL DD RESOLVE-AUTH CHAR( 2) DEF(RESOLVE) POS(33) INIT(X'0000');

DCL SPCPTR .MAKPTR INIT(MAKPTR);
DCL DD MAKPTR CHAR(16) BDRY(16);
  DCL SPCPTR .SPCPTR DEF(MAKPTR) POS(1); /* fabricated space pointer */
  DCL SYSPTR .SYSPTR DEF(MAKPTR) POS(1);
  DCL DD PTR-ADDRESS CHAR(8) DEF(MAKPTR) POS(9); /* address to make pointer from */

DCL OL MIMAKPTR (.MAKPTR) ARG;
DCL SYSPTR .MIMAKPTR;
```

Because the data portion of the dump space is accessed through its 64-bit address instead of via a normal MI-pointer, we need our trusted program **MIMAKPTR** that fabricates a pointer from an address:

```
RESOLVE-TO-HELPER-PROGRAM:
  CPYBLA RESOLVE-TYPE, X'0201';
  CPYBLAP RESOLVE-NAME, "MIMAKPTR", " ";
  RSLVSP .MIMAKPTR, RESOLVE, *, *;

DCL DD SAVE-POINTER CHAR(16) BDRY(16);
DCL SYSPTR .SAV-FILE DEF(SAVE-POINTER) POS(1);
```

Of course, we need a system pointer to the save file as well:

```
RESOLVE-TO-SAVE-FILE:
  CPYBLA RESOLVE-TYPE, X'1901';
  CPYBLAP RESOLVE-NAME, SAVE-FILE, " ";
  RSLVSP .SAV-FILE, RESOLVE, *, *;

GET-ASSOCIATED-SPACE:
  SETSPFPF .SPCPTR, .SAV-FILE;
```

To get a pointer to the primary associated space using the **SETSPFPF** instruction our program needs to have the *system state* attribute, but it needs that anyway because the data we are going to access is in the system domain. We can now base a storage area on the .SPCPTR space pointer:

```
DCL DD STORAGE CHAR(16000000) BAS(.SPCPTR);
DCL DD OFFSET BIN(4);
```

We can now get the address of the dump space object and turn it into a space pointer:

```
GET-DUMP-SPACE:
  ADDSPP .SPCPTR, .SPCPTR, H'0280'; /* DUMP SPACE */
  CPYBLAP PTR-ADDRESS, STORAGE(9:5), X'00';
  CALLX .MIMAKPTR, MIMAKPTR, *;
```

There is no “normal” (system supplied or supported) way of doing this as we wish to access the “encapsulated” (and thus in-accessible) part of the object, but MIMAKPTR does the job just fine. One could argue that this is not MI-programming at all, and one would be correct; it is, however, superb *machine-level* programming.

We can now get the address of the data portion and turn that into a space pointer:



```

GET-DATA-ADDRESS:
  ADDSPP      .SPCPTR, .SPCPTR, H'1000';
  CPYBLA      PTR-ADDRESS, STORAGE(1:8);
  CALLX      .MIMAKPTR, MIMAKPTR, *;

```

We can advance to the second page of the data:

```

POINT-TO-SECOND-PAGE:
  ADDSPP      .SPCPTR, .SPCPTR, H'1000';

```

Since we don't really care about the system model number (which is the only thing stored on the second page), we skip one more page down to the (first) dump descriptor:

```

POINT-TO-DUMP-DESCRIPTOR:
  ADDSPP      .DESCR, .SPCPTR, H'1000';
  CMPBLA(B)   DESCR-DUMP-ID, "QSRDSSPC.1"/NEQ(DONE);
  MULT       .DESCR-SIZE, DESCR-UNITS, 512;

DCL SPCPTR .DESCR;
DCL DD DESCR-SIZE BIN(4);
DCL DD DUMP-DESCRIPTOR CHAR(4096) BAS(.DESCR);
DCL DD DESCR-FFFFFFFF CHAR(4) DEF(DUMP-DESCRIPTOR) POS( 1);
DCL DD DESCR-DUMP-ID CHAR(10) DEF(DUMP-DESCRIPTOR) POS( 5);
DCL DD DESCR-UNITS      BIN(4) DEF(DUMP-DESCRIPTOR) POS(205);

```

As a sanity check, we make sure that the dump descriptor has the "QSRDSSPC.1" name. The size (in bytes) of the dump descriptor can now be calculated as 512 times the number of *units* in the descriptor. We save the pointer to the description in .DESCR for later use. The catalog is in the next page (add x'1000'):

```

POINT-TO-DUMP-CATALOG:
  ADDSPP      .SPCPTR, .DESCR, H'1000';
  CMPBLA(B)   DUMP-TYPE, X'19DB'/NEQ(DONE);

```

Again, we check that we've got what we expect. The type must be x'19DB'. Below is the format of the catalog. Note how we, arbitrarily, support a maximum of 512 catalog entries. Alternatively, we could have declared the entry array with only one element and then used the **OVIRPGATR** MI-instruction to override subscript checking:

```

DCL DD CUR-ENTRY BIN(4);
DCL DD CATALOG CHAR(16000000) BAS(.SPCPTR);
DCL DD DUMP-TYPE      CHAR(2) DEF(CATALOG) POS( 35);
DCL DD DUMP-CONTEXT   CHAR(30) DEF(CATALOG) POS(259);
DCL DD DUMP-CONTEXT-TYPE CHAR(2) DEF(CATALOG) POS(289);
DCL DD DUMP-TIMESTAMP CHAR(8) DEF(CATALOG) POS(291);
DCL DD DUMP-SERIAL-NBR CHAR(7) DEF(CATALOG) POS(310);
DCL DD DUMP-NBR-ENTRIES BIN(4) DEF(CATALOG) POS(329);

DCL DD DUMP-ENTRY (512) CHAR(151) DEF(CATALOG) POS(337) AEO(151);
DCL DD DUMP-OBJ-NAME(512) CHAR(30) DEF(CATALOG) POS(337) AEO(151);
DCL DD DUMP-OBJ-TYPE(512) CHAR(2) DEF(CATALOG) POS(367) AEO(151);
DCL DD DUMP-OBJ-USER(512) CHAR(30) DEF(CATALOG) POS(369) AEO(151);
DCL DD DUMP-OBJ-MORE(512) BIN(4) DEF(CATALOG) POS(408) AEO(151);
DCL DD DUMP-OBJ-INFO(512) BIN(4) DEF(CATALOG) POS(412) AEO(151);

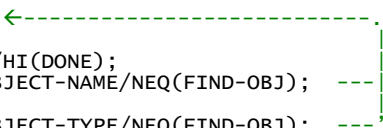
```

Finding the object entails searching through the catalog, comparing object name and type with name and type of the objects listing in the catalog:

```

  CPYNV      CUR-ENTRY, 0;
FIND-OBJ:
  ADDN(S)     CUR-ENTRY, 1;
  CMPNV(B)    CUR-ENTRY, DUMP-NBR-ENTRIES/HI(DONE);
  CMPBLA(B)    DUMP-OBJ-NAME(CUR-ENTRY), OBJECT-NAME/NEQ(FIND-OBJ);
CHECK-TYPE:
  CMPBLA(B)    DUMP-OBJ-TYPE(CUR-ENTRY), OBJECT-TYPE/NEQ(FIND-OBJ);

```



Once we have located the object, we can extract the information. Since our main goal here was the *find* the information, we'll just show a crude digest of the information (digits and upper-case letters):

```

DCL SPCPTR .HEADER;
POINT-TO-OBJ-INFORMATION:
  ADDSPP      .HEADER, .SPCPTR, H'0100';

```

First the creation-related information (if any):

```

        CMPNV(B)      DUMP-OBJ-INFO(CUR-ENTRY), 0/EQ(=+4);
HAVE-OBJECT-INFO:
        ADDSPP        .INFO, .HEADER, DUMP-OBJ-INFO(CUR-ENTRY);
        CPYNV         SIZE, 214;
        CALLI         SHOW-INFO, *, .SHOW-INFO;:

```

Then the ‘more’ information (if any):

```

        CMPNV(B)      DUMP-OBJ-MORE(CUR-ENTRY), 0/EQ(=+4);
HAVE-OBJECT-MORE-INFO:
        ADDSPP        .INFO, .HEADER, DUMP-OBJ-MORE(CUR-ENTRY);
        CPYNV         SIZE, 2000;
        CALLI         SHOW-INFO, *, .SHOW-INFO;:
DONE:
        RTX          *;

```

The following simple routine constructs a message with only “printable” characters. Up to **SIZE** bytes are examined:

```

DCL SPCPTR .INFO;
DCL DD INFO CHAR(4096) BAS(.INFO);
DCL DD CHAR CHAR(1);
DCL DD WHERE BIN(2);
DCL DD SIZE BIN(2);
DCL DD FROM BIN(2);
DCL DD TO BIN(2);

DCL INSPTR .SHOW-INFO;
ENTRY      SHOW-INFO INT;
        CPYBREP      MSG-TEXT, " ";
        CPYNV        FROM, 0;
        CPYNV        TO, 0;

SHOW-NEXT-CHAR:
        CMPNV(B)      FROM, SIZE/NLO(SHOW-IT);
        ADDN(S)        FROM, 1;
        CPYBLA        CHAR, INFO(FROM:1);
        VERIFY(B)      WHERE, CHAR, "ABCDEFGH IJKLMNOPQRSTUVWXYZ0123456789"
                        /NZER(SHOW-NEXT-CHAR);

        CMPNV(B)      TO, 70/NLO(SHOW-IT); /* 70 is size of MSG-TEXT */
        ADDN(S)        TO, 1;
        CPYBLA        MSG-TEXT(TO:1), CHAR;
        B             SHOW-NEXT-CHAR;

SHOW-IT:
        CALLI         SHOW-MESSAGE, *, .SHOW-MESSAGE;
        B             .SHOW-INFO;

```

```
%INCLUDE SHOWMSG
```

## Walk Descriptors in Save File, **MIWLKSAV**

The number of objects saved is the same as the number of entries in the dump catalog. Each object that is saved in the save file has its own dump descriptor. The binary object follows on the next save file page:

```

006000  FFFFFFFF D4C9C1C4 E2E4D4C9 40404040    ...MIADSUMI
006010  40404040 40404040 40404040 40404040
006020  40400201 00000050 00000000 00000000    .....&.....
006030  00000000 00000000 00000000 00000000    .....
006040  00000000 00000260 0000003F 00000200    .....-.....
006050  36003600 36000000 00000000 00000000    .....
006060  00200000 00000004 00000000 00000000    .....
006070  00000000 00000000 00000000 00000000    .....
006080  00000000 00000000 00000000 00000000    .....
006090  00000000 0000D361 C440D6C2 D1C5C3E3    .....L/D OBJECT
0060A0  40C4C5E2 C3D9C9D7 E3D6D940 40400000    DESCRIPTOR ..
0060B0  00000000 00000000 0000C4C9 E2D20014    .....DISK..
0060C0  828A8EB8 B36E0000 0000FF03 00000040 b«p½->..... ← size in units
. . .
007000  00010018 00900001 392FCB44 FA000000    .....°.....ðà³... ← next page
007010  30010300 00000000 17675E8E 26001000    .....A;p....
007020  80000201 D4C9C1C4 E2E4D4C9 40404040    Ø...MIADSUMI
007030  40404040 40404040 40404040 40404040
. . .

```

The following little program, **MIWLKSAV**, “walks” all objects in the save file and locates each binary object. Its structure is very similar to the **MIOBJSAV** program. Call the program with the name of a save file (on the

library list) as the sole parameter. We present the program without further comments. Its working should be clear by now. A break point, **BRK "1"**, is placed at the point where an object is available in the STORAGE data item:

```

DCL SPCPTR .P1 PARM; DCL DD SAVE-FILE CHAR(10) BAS(.P1);

DCL OL PARMS(.P1) PARM EXT MIN(1);
.
.
DCL DD SAVE-POINTER CHAR(16) BDRY(16);
DCL SYSPTR .SAV-FILE DEF(SAVE-POINTER) POS(1);

DCL SPCPTR .MAKPTR INIT(MAKPTR);
DCL DD MAKPTR CHAR(16) BDRY(16);
DCL SPCPTR .SPCPTR DEF(MAKPTR) POS(1);
DCL SYSPTR .SYSPTR DEF(MAKPTR) POS(1);
DCL DD PTR-ADDRESS CHAR(8) DEF(MAKPTR) POS(9);

DCL OL MIMAKPTR (.MAKPTR) ARG;
DCL SYSPTR .MIMAKPTR;

DCL DD STORAGE CHAR(16000000) BAS(.SPCPTR);

DCL SPCPTR .DESCR;
DCL DD DESCR-SIZE BIN(4);
DCL DD DUMP-DESCRIPTOR CHAR(4096) BAS(.DESCR);
DCL DD DESCR-FFFFFFFF CHAR(4) DEF(DUMP-DESCRIPTOR) POS( 1);
DCL DD DESCR-DUMP-ID CHAR(10) DEF(DUMP-DESCRIPTOR) POS( 5);
DCL DD DESCR-UNITS BIN(4) DEF(DUMP-DESCRIPTOR) POS(205);

DCL DD ENTRIES BIN(4);
DCL DD CATALOG CHAR(16000000) BAS(.SPCPTR);
DCL DD DUMP-NBR-ENTRIES BIN(4) DEF(CATALOG) POS(329);

ENTRY * (PARMS) EXT;
RESOLVE-TO-HELPER-PROGRAM:
. . .

RESOLVE-TO-SAVE-FILE:
CPYBLA RESOLVE-TYPE, X'1901';
CPYBLAP RESOLVE-NAME, SAVE-FILE, " ";
RSLVSP .SAV-FILE, RESOLVE, *, *;

GET-ASSOCIATED-SPACE:
SETSPFPF .SPCPTR, .SAV-FILE;

GET-DUMP-SPACE:
ADDSP .SPCPTR, .SPCPTR, H'0280'; /* DUMP SPACE */
CPYBLAP PTR-ADDRESS, STORAGE(9:5), X'00';
CALLX .MIMAKPTR, MIMAKPTR, *;

GET-DATA-ADDRESS:
ADDSP .SPCPTR, .SPCPTR, H'1000';
CPYBLA PTR-ADDRESS, STORAGE(1:8);
CALLX .MIMAKPTR, MIMAKPTR, *;

POINT-TO-SECOND-PAGE:
ADDSP .SPCPTR, .SPCPTR, H'1000';

POINT-TO-DUMP-DESCRIPTOR:
ADDSP .DESCR, .SPCPTR, H'1000';
CMPBLA(B) DESCR-DUMP-ID, "QSRDSSPC.1"/NEQ(DONE);

POINT-TO-DUMP-CATALOG:
ADDSP .SPCPTR, .DESCR, H'1000';
CMPBLA(B) DUMP-TYPE, X'19DB'/NEQ(DONE);
CPYNV(B) ENTRIES, DUMP-NBR-ENTRIES/ZER(DONE);

NEXT-DUMP-DESCRIPTOR:
MULT DESCR-SIZE, DESCR-UNITS, 512;
ADDSP .DESCR, .DESCR, DESCR-SIZE;
CMPBLA(B) DESCR-FFFFFFFF, X'FFFFFFFF'/NEQ(DONE); ← see below

POINT-TO-BINARY-OBJECT:
ADDSP .SPCPTR, .DESCR, H'1000';
/* 'STORAGE' NOW CONTAINS THE OBJECT */
BRK "1";
SUBN(SB) ENTRIES, 1/POS(NEXT-DUMP-DESCRIPTOR);

DONE:

```

RTX                   \* ;

Well, maybe one comment: I have seen a few cases where the number of entries given in the catalog was *one higher* than the number of dump descriptors actually present. Since a dump descriptor starts with a binary value of all ones (DESCR-FFFFFFF) I use that as a sanity check and stop if the expected data word is not found.

## Modifying a Save File

Now that we can “walk” a save file and can find the saved objects and various pieces of information about them, we turn to the subject of *changing* some of information found. Why would you change data stored in a save file? Here are some reasons:

- Promotion management: Programs may be developed in a test environment and should be “promoted” to production. In particular, you may wish to change the libraries, where the programs and modules reside, to reflect the production environment rather than the test environment. Surprisingly, this cannot be done using standard OS/400 means (*e.g.* see the help text for MODULE in the UPDPGM command).
- Creation of unique save files: Maybe you are a software vendor and you wish to embed customer specific information (*e.g.* a machine serial number) in a data area or even in a program (where it cannot be changed by the customer!). You do not want to change your “master version” from which the save file is produced.
- Cracking: You do not have access (or authority) to System Service Tools, or you wish to avoid your activity being logged in the Alter Log or in an audit journal. (I did not use the honorable term “hacking” for this dubious activity).
- Removal of Own Identity: A save file contains the serial number, system name, owner (or developer) user profile name, and other information that you do not wish to expose to your customers.

## Save File Integrity? NOT

We have already seen that when a save file is moved to an external medium (or is accessed as a file), two different kinds of checksums are calculated. One checksum for each block and one overall checksum over the entire save file. The block-checksums are obviously not present in the save file object (as there is no room for a checksum in each block), but the overall checksum could well be stored somewhere in the save file object:

```
Address 318A4197C7 000000
000000 00010010 00818000 318A4197 C7000000  . . . . a0 . . « . pG . .
000010 C0010000 00000000 1918266A 12000020  { . . . . . . . . . . . .
000020 80001390 E2C1E5C5 C6C9D3C5 4040E3C8  0 . . ° SAVEFILE TH  ← name
000030 C5D3C9C2 40404040 F0F14040 40404040  ELIB 01
000040 40408000 00000FE0 00000520 00403000  0 . . . . \ . . . . .
```

At offset x'230' we find the following:

```
000230 00000000 00000000 00000000 00000000  . . . . .
000240 00000000 00000000 00000000 00000000  . . . . .
```

At first sight, the string of zeroes here is not too remarkable. It must be said that this save file was “virgin” in the sense of being just created and populated with objects. Now, if you FTP the save file to another system or create a duplicate of it (CRTDUPOBJ), you’ll see those zeroes change (in *both* the original file and the copy) to something like this:

```
000230 00000000 00000000 82C979D6 B1C78000  . . . . . bI`OfG0.  ← timestamp
000240 BEA80635 9FB91CF6 6DB43B5E F1F8BBF0  `¿ . . ¢% . 6_@ . ;18]0  ← overall checksum
```

A timestamp has been inserted and a checksum has been calculated. If you now change anything in the save file, the checksum will be incorrect and you *invalidate* the save file (actually the dump space). If you try to duplicate the save file again, you get an error like:

```
Invalid dump data insertion for dump space SAVEFILE LSVALGAARD01
```

It would seem that save files are protected against tampering with. But it really isn't so, because you can make the changes before the checksum is calculated or simply zero-out the timestamp and the checksum. The security flaw here is that a save file without a checksum is accepted as valid (probably to be backward compatible). It is also possible to re-calculate the checksum as the algorithm is readily re-engineered, but there is really no need: just set the checksum to zero.

## ***Single-Level Store? Maybe Not Always***

So, you have decided to make some changes to a save file, perhaps to change the source library for a program to reflect the fact that the program is being promoted to production (and the sources copied over to production as well). You make the change using an enhanced version of MIOBJSAV, restore the program over in production and do a DSPPGM to check that the source library changed as desired. Most of the time you'll find that it has *not* changed. You go back to the save file and check; in the save file object the source information shows as changed. What is going on here? Was the benefit of the single-level store not that data is only kept in one place as far as programs above the MI were concerned? In fact, an object may reside both in memory and on the disk(s). Storage Management's job is to ensure a consistent image of the address space. If you change something in memory the change will eventually be transferred to the disk (if not before then at a normal power-down), but for obvious performance reasons memory and disk are not "in sync" at all times.

The solution to the "vanishing changes" puzzle is that the save/restore programs do not access a save file as a set of objects, but as a file on a physical medium. Historically you saved files to diskette or tape, *i.e.* physical media, so maybe it is not surprising that the save/restore operations writes/reads directly to/from the disk without going through an "object" in memory. If you make the changes with SST you will find that they make it to the disk. This is because SST alters the disk image rather than the memory image (also the reason why SST can alter write-protected programs).

## **"Flushing" a Save File to Disk**

CRTDUPOBJ, on the other hand, does create a new object on the disk from the object in memory, so here is a way of "flushing" the object in memory to disk. The steps to take in order to change an object via a save file (which may be the only option if the object is write-protected) are then:

```

CRTSAVF FILE(X)
SAVOBJ OBJ(AAA) LIB(LLL) DEV(*SAVF) SAVF(X)
CALL CHGSAV PARM(. . .) ← this is your program to change a save file
CRTDUPOBJ OBJ(X) OBJLIB(LLL) OBJTYPE(*FILE) NEWOBJ(XX) DATA(*YES)
RSTOBJ OBJ(AAA) SAVLIB(LLL) DEV(*SAVF) SAVF(XX)
DLTF FILE(XX)
DLTF FILE(X)

```

The whole process can be neatly automated.

## ***Changing Module Information, MICHGMOD***

With what we have learned, we can attack a somewhat bigger problem: the "promotion" problem. We are given a save file containing an ILE program that we wish to promote to production. The program may contain scores of modules. We'll create a program, **MICHGMOD**, to change information about a module bound into the ILE program. We call MICHGMOD for each module we wish to change. When done, we can then restore the ILE program to the production library. We want to be able to *debug* the module with the debugger using the source code at its new location. To locate a module, we need the following input parameters:

- Name of save file
- Name of library containing the save file
- Name of program
- Name of library containing the program
- Type of program (\*PGM or \*SRVPGM)
- Name of module
- Name of library containing the module

We wish to change any or all of the following:

- Name of library containing the module
- Name of source file containing the module source member
- Name of source library containing the source file

If no change is wanted for any of these the parameter value should be \*SAME. To make it easier to manage (e.g. to prompt) so many parameters, a command, **CHGMINF**, is called for:

```

CMD      PROMPT('Fix module in save file')
PARM     KWD(SAVF) TYPE(*CHAR) LEN(10) PROMPT('Save +
          file')
PARM     KWD(SAVFLIB) TYPE(*CHAR) LEN(10) +
          PROMPT('Save file library')
PARM     KWD(PGM) TYPE(*CHAR) LEN(10) PROMPT('Program')
PARM     KWD(PGMLIB) TYPE(*CHAR) LEN(10) +
          PROMPT('Program library')
PARM     KWD(PGMTYPE) TYPE(*CHAR) LEN(10) RSTD(*YES) +
          DFT(*PGM) VALUES(*PGM *SRVPGM) +
          PROMPT('Program type')
PARM     KWD(MOD) TYPE(*CHAR) LEN(10) PROMPT('Module')
PARM     KWD(MODLIB) TYPE(*CHAR) LEN(10) +
          PROMPT('Module library')
PARM     KWD(NEWLIB) TYPE(*CHAR) LEN(10) DFT(*SAME) +
          PROMPT('Module new library')
PARM     KWD(NEWSFIL) TYPE(*CHAR) LEN(10) DFT(*SAME) +
          PROMPT('Module new source file')
PARM     KWD(NEWSLIB) TYPE(*CHAR) LEN(10) DFT(*SAME) +
          PROMPT('Module new source library')
PARM     KWD(RETURN) TYPE(*CHAR) LEN(10) RTNVAL(*YES) +
          PROMPT('Return value')

```

Create the command as follows (making sure the command-processing program is **MICHGMOD**):

```
====> CRTCMD CMD(*CURLIB/CHGMINF) PGM(MICHGMOD) ALLOW(*BPGM *IPGM)
```

The MICHGMOD program starts out like the two other save file programs we've developed in this chapter. When the DUMP-CATALOG has been located, it then scans the catalog for the program of the proper type:

```

POINT-TO-DUMP-CATALOG:
  ADDSPP      .SPCPTR, .DESCR, H'1000';
  CMPBLA(B)   DUMP-TYPE, X'19DB'/NEQ(DONE);
  CPYBLAP     RETURN-VALUE, "PGMLNF", " ";
  CMPBLA(B)   DUMP-CONTEXT, PROGRAM-LIBRARY/NEQ(DONE);

  CPYBLAP     RETURN-VALUE, "PGMNF", " ";
  CPYV        CUR-ENTRY, 0;
FIND-PGM:
  ADDN(S)     CUR-ENTRY, 1;
  CMPNV(B)    CUR-ENTRY, DUMP-NBR-ENTRIES/HI(DONE);
  CMPBLA(B)   DUMP-OBJ-NAME(CUR-ENTRY), PROGRAM-NAME/NEQ(FIND-PGM);
CHECK-TYPE:
  CMPBLA(B)   DUMP-OBJ-TYPE(CUR-ENTRY), HEX-TYPE /NEQ(FIND-PGM);

```

HEX-TYPE is \*PGM translated into x'0201' or \*SRVPGM translated into x'0203'.

```

FIND-PROGRAM-DESCRIPTOR:
  MULT        DESCR-SIZE, DESCR-UNITS, 512;
  ADDSPP      .DESCR, .DESCR, DESCR-SIZE;
  SUBN(SB)    CUR-ENTRY, 1/POS(FIND-PROGRAM-DESCRIPTOR);

POINT-TO-PROGRAM-OBJECT:
  MULT        DESCR-SIZE, DESCR-UNITS, 512;
  CPYV        OFFSET, 0;
FIND-EPA-HDR:
  ADDN(S)     OFFSET, H'1000';
  ADDSPP      .PGM, .DESCR, OFFSET;

POINT-TO-EPA-HEADER:
  ADDSPP      .SPCPTR, .PGM, 32;
  CMPBLAP(B)  PROGRAM-EPA-NAME, PROGRAM-NAME, " "/NEQ(FIND-EPA-HDR);

```

Once we have located the program object header, we can use what we learned in chapter 15 about the structure of program to locate the module:

```

POINT-TO-PROGRAM-HEADER:
  CPYBRAP      OFFSET, PROGRAM-HDR-ADDR(6:3), X'00';
  ADDSPP       .SPCPTR, .PGM, OFFSET;

POINT-TO-MODULE-TABLE:
  CPYBRAP      OFFSET, MODULE-TBL-ADDR(6:3), X'00';
  ADDSPP       .MOD-TBL, .PGM, OFFSET;

  CPYBLAP      RETURN-VALUE, "MODNF", " ";
  CPYNV        CUR-MODULE-NBR, 0;
LOOK-AT-MODULES:
  ADDN(S)      CUR-MODULE-NBR, 1;
  CMPNV(B)     CUR-MODULE-NBR, NBR-OF-MODULES/HI(DONE);
  CPYBRAP      OFFSET, MOD-HDR-OFFSET(CUR-MODULE-NBR), X'00';
  ADDSPP       .MOD-HDR, .PGM, OFFSET;

LOOK-AT-MODULE-VERSION:
  CPYBRAP      OFFSET, MOD-VERSION-ADDRESS(6:3), X'00';
  ADDSPP       .SPCPTR, .PGM, OFFSET;
  CMPBLAP(B)   MOD-NAME, MODULE-NAME, " "/NEQ(LOOK-AT-MODULES);

```

In this version we did not check the **MODULE-LIBRARY** against **MOD-CONTEXT**, but we *could* have:

```

HAVE-THE-MODULE:
  CMPBLAP(B)   MODULE-NEW-LIBRARY, "*SAME", " "/EQ(=+2);
  CPYBLAP      MOD-CONTEXT, MODULE-NEW-LIBRARY, " ";;

  CPYBLAP      RETURN-VALUE, "BNASNF", " ";
  CPYNV        OFFSET, 0;
  CMPBLAP(B)   MODULE-NEW-SRC-FILE, "*SAME", " "/NEQ(FIND-BNAS);
  CMPBLAP(B)   MODULE-NEW-SRC-LIBRARY, "*SAME", " "/NEQ(FIND-BNAS);
  B            OK;

```

In Chapter 38, we'll investigate how and where source file and debug information are stored. Here we show how to find that information and change it. First we look for the **"BNAS"** signature:

```

FIND-BNAS:      /* BOUND PROGRAM ASSOCIATED SPACE */
  ADDSPP        .SPCPTR, .DESCR, OFFSET;
  CMPBLA(B)     BNAS-SIGNATURE, X'02039301C2D5C1E2'/EQ(HAVE-BNAS);
  ADDN(S)       OFFSET, 16;
  CMPNV(B)      OFFSET, DESCR-SIZE/LO(FIND-BNAS), NLO(DONE);

HAVE-BNAS:
  ADDSPP        .SPCPTR, .SPCPTR, BNAS-MODULES-OFFSET;
  CMPBLA(B)     BNAS-SIGNATURE, X'0100000000010002'/EQ(HAVE-MODULES);
  CMPNV(B)      OFFSET, DESCR-SIZE/LO(HAVE-BNAS), NLO(DONE);

HAVE-MODULES:
  SETSPP        .MODULE, BNAS-MODULE(CUR-MODULE-NBR);
  CMPBLAP(B)    MODULE-NEW-SRC-FILE, "*SAME", " "/EQ(=+2);
  CPYBLA        BNAS-SOURCE-FILE, MODULE-NEW-SRC-FILE;;
  CMPBLAP(B)    MODULE-NEW-SRC-LIBRARY, "*SAME", " "/EQ(=+2);
  CPYBLA        BNAS-SOURCE-LIBRARY, MODULE-NEW-SRC-LIBRARY;;

```

Then we check to see if a "High-Level Language symbol Table" is present:

```

  CPYNV        THE-MODULE-NBR, 0;
FIND-HLL:      /* HLL SYMBOL TABLE */
  CMPNV(B)     OFFSET, DESCR-SIZE/NLO(OK);
  ADDN(S)      OFFSET, 16;
  ADDSPP       .SPCPTR, .DESCR, OFFSET;
  CMPBLA(B)    HLL-SIGNATURE, "HLL Symbol Table"/NEQ(FIND-HLL);
  ADDN(S)      THE-MODULE-NBR, 1;
  CMPNV(B)     THE-MODULE-NBR, CUR-MODULE-NBR /LO (FIND-HLL);

```

The debugger looks for something called a **"FILE DESCRIPTOR"**, so we look for that too:

```

FIND-FILE:
  ADDSPP        .SPCPTR, .DESCR, OFFSET;
  CMPBLA(B)     FILE-SIGNATURE, "FILE DESCRIPTOR"/EQ(HAVE-FILE);
  ADDN(S)       OFFSET, 16;
  CMPNV(B)      OFFSET, DESCR-SIZE/LO(FIND-FILE), NLO(OK);

HAVE-FILE:
  CMPBLAP(B)    MODULE-NEW-SRC-FILE, "*SAME", " "/EQ(=+2);
  CPYBLA        FILE-SOURCE-FILE, MODULE-NEW-SRC-FILE;;
  CMPBLAP(B)    MODULE-NEW-SRC-LIBRARY, "*SAME", " "/EQ(=+2);
  CPYBLA        FILE-SOURCE-LIBRARY, MODULE-NEW-SRC-LIBRARY;;

```

Sometimes there are *two* file descriptors back-to-back; the debugger looks for the second one, and so do we:

```
ADDN(S)      OFFSET, 80;  
CMPBLA(B)    FILE-SIGNATURE2, "FILE_DESCRIPTOR"/EQ(FIND-FILE);
```

```
OK:  
CPYBLAP      RETURN-VALUE, " ", " ";
```

The RETURN-VALUE will be all blanks if the module information was changed successfully, otherwise a short error code is returned:

```
DONE:  
RTX          *;
```

### ***Save Files for CISC Architecture***

The format of save files for CISC AS/400s is very similar to that for RISC-based machines. The main difference is that the page size on CISC machines is 512 bytes versus 4096 bytes for RISC.



## Chapter 28

### Recursion, Entropy, and the End of the World

#### ***Recursive Procedures***

A recursive program (module, procedure, ...) is a program that directly or indirectly calls itself. The indirect calls can be important as well. A classical case is a “print” function that evaluates a parameter that is a function calling a program using “print” for debugging. The factorial function  $n!$  for a non-negative integer  $n$  is often defined recursively as  $n! = n(n-1)!$ . It is clear that recursion has to stop eventually; the function must “bottom out” sooner or later. For the factorial, this happens when the factor becomes 1. To use the factorial as an example of practical recursion is, however, extremely silly, as the factorial is much better, simpler, and quicker calculated by iteration rather than by recursion.

At the MI-level, the AS/400 supports recursion directly in a straightforward manner, but you can use recursive techniques with any language: old RPG, COBOL, and even the lowest level assembler language. Recursion involves certain machinations. The issue is just who does the machination? The system or you. Recursive procedures were introduced into high-level languages by the famous ALGOL-60 report, that without any fanfare simply stated that “the procedure body is inserted in place of the procedure statement, and executed” and that “any occurrence of the procedure identifier within the body of the procedure [...] denotes activation of the procedure”, implying recursion as a matter of course.

Recursion is most useful with complicated data structures that nest other structures that in turn include others, and so on. Such structures occur in parsing, evaluation of expressions, “traversals” of lists and tree-structures, and such-like “computer science” applications. But even in traditional business applications you find situations that are best handled by recursion, *e.g.* involving a master file, where each record can contain (or point to) a set of detail records, which in turns point to details of the details and so on to great depth. A problem for an author like me in this regard is that it is hard (nay, quite impossible) to work through realistic business applications to show you how recursion works and how you can benefit from it.

There is, however, a class of problems of the simpler nature, that, indeed, are best solved by iteration, but where it is not evident how to do the iteration. A recursive version is often quite straightforward to set up. Transforming recursion into iteration is a purely mechanical process if you feel that you need the extra speed that an iterative version gives you (those “machinations” do carry *some* cost, although often much smaller than thought). In this chapter we’ll look at two such small, but instructive, problems. First, we noted in Chapter 22 that adding several numbers together destroys a lot of the entropy contained in them. If the sum of 5 non-negative numbers is 10, there are 1001 different ways they could sum to 10. How do we calculate that number, 1001? The second problem, we’ll attack, is the venerable (in fact: recurrent!) problem of the “Tower of Hanoi”.

#### ***Summing of Integers***

A  $k$ -combination of items from a set consists of  $k$  members from the set without regard to their order. How many distinct  $k$ -combinations,  $Q(n, k)$ , can I make from a set having  $n$  members when I’m allowed to select members more than once? Note that  $k$  can be larger than  $n$ , because we are “re-using” elements. This is known as selection with replacement (from the metaphor of drawing marbles from an urn and replacing them back into the urn after each drawing of a single marble). A nice way of finding the answer to this rather tricky problem is to note, that it is the same as the number of integer solutions  $(a_1, a_2, \dots, a_k)$  to the relations  $1 \leq a_1 \leq a_2 \leq \dots \leq a_k \leq n$ . (Imagine that the marbles were marked with little numbers: 1, 2, ...,  $n$ , and when we pick  $k$  of them we lay them out in order of their number. The equal condition in “ $\leq$ ” refers to the fact that we can pick the same marble several times.)

The first relation ( $1 \leq a_1$ ) is the same as  $0 < a_1$ : if  $a_1$  is greater or equal to 1, it is certainly greater than 0. The second relation ( $a_1 \leq a_2$ ) is the same as  $a_1 < a_2 + 1$  for the same reason. The third relation ( $a_2 \leq a_3$ ) is the same as  $a_2 < a_3 + 1$ , or adding one on both sides:  $a_2 + 1 < a_3 + 2$ . Continuing like this, we can state the relations as  $0 < a_1 < a_2 + 1 < a_3 + 2 < \dots < a_k + k - 1 < n + k$ . Renaming  $a_j + j - 1$  to  $b_j$ , we can rewrite the relations as:  $0 < b_1 < b_2 < b_3 < \dots < b_k < n + k$ , where obviously all the  $b$ ’s are distinct. The number of

solutions to that relation is then clearly the number of times I can pick  $k$  items *without* replacement (because they are distinct, no two will bear the same number) from the set  $\{1, 2, 3, \dots, n + k - 1\}$ , where we just translated the  $b$ 's back into  $a$ 's and then dropped the  $a$  as just the mental crutch it was. Isn't pure symbol manipulation wonderful?

Now, the number of times,  $P(n, k)$ , that I can pick  $k$  items out of  $n$  without replacement is the famous “ $n$  choose  $k$ ” function. We derive an expression for that function as follows. A *permutation* of  $n$  objects is an arrangement of  $n$  distinct objects in a row. How many permutations of  $n$  objects are possible? There are  $n$  ways to choose the first object, and  $n - 1$  ways to choose a different object. Similarly, there are  $n - 2$  choices for the third object, and so on. In general, if  $p_{nk}$  is the number of ways in which to choose  $k$  objects out of  $n$  and to arrange them in a row, we have:  $p_{nk} = n(n-1)(n-2) \dots (n-k+1)$ . Therefore, the total number of permutations is  $p_{nn} = n(n-1)(n-2) \dots (1) = n!$ . So, there are  $p_{nk}$  ways to choose the first  $k$  objects for a permutation; and every  $k$ -combination appears  $k!$  times in these arrangements, since each combination appears in all its permutations (of which, with  $k$  elements, there are  $k!$ ). Hence, the total number of combinations becomes  $P(n, k) = p_{nk} / k! = n! / k! (n-k)!$ . Note that in standard mathematical texts, the multiplication sign implied by omitting it has a higher precedence than the division sign; i.e.  $ab/cd$  means  $(a \cdot b)/(c \cdot d)$ , so,  $P(n, k)$  is usually written as  $P(n, k) = n! / k!(n - k)!$

We can then finally write

$$\begin{aligned} Q(n, k) &= P(n+k-1, k) = (n+k-1)! / k! (n+k-1-k)! \\ &= (k + (n-1))! / k! (n-1)! \end{aligned}$$

After this eye-glazing-over preliminary, we turn to the problem proper. Let's start with a small example: take 3 non-negative integers and ask in how many ways they can add to 4. Clearly, no value can be higher than 4 to begin with. We can make a little table showing all combinations of values that sum to 4. There are 15 combinations:

1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	“Sticks”		
0	0	4	0000	0000	1111
0	1	3	0000	0001	0111
0	2	2	0000	0011	0011
0	3	1	0000	0011	0001
0	4	0	0000	1111	0000
1	0	3	0001	0000	0111
1	1	2	0001	0001	0011
1	2	1	0001	0011	0001
1	3	0	0001	0111	0000
2	0	2	0011	0000	0011
2	1	1	0011	0001	0001
2	2	0	0011	0011	0000
3	0	1	0111	0000	0001
3	1	0	0111	0001	0000
4	0	0	1111	0000	0000

In the “sticks” column we show the same three numbers as “sticks” rather than arithmetic numbers. Consider the set of “stick numbers”  $\{100, 010, 001\}$ . If I pick 4 elements from this set *with* replacement, I get exactly 4 “sticks” (since each element contains precisely one stick). The total number of distinct 4-combinations that I can select with replacement from a set of 3 members is  $Q(2, 4) = 15$ , which is just the number of combinations in the table. Since both the  $k$ -combinations are distinct and the members of the “sticks” columns are distinct, we can match them one for one in any way we wish. The argument is quite general and we have thus proved that **the number of ways  $n$  numbers can add to  $k$  is the number of distinct  $k$ -combinations out of a set with  $n$  members**. In other words: the number of solutions in non-negative integers to  $k_1 + k_2 + k_3 + \dots + k_n = k$  is  $Q(n, k)$ . This is the result we were looking for.

These (arcane?) considerations actually have wide applications, and appropriately enough especially in the context of entropy. The entropy concept comes from Thermodynamics, where a system is said to have maximum entropy if it is in *thermal equilibrium*. For such systems the important theorem of “Equipartition of Energy” applies.

## Equipartition of Energy

The theorem of equipartition of energy states that molecules in thermal equilibrium have the same average energy associated with each independent degree of freedom of their motion. The Maxwell-Boltzmann distribution is the classical distribution function for distribution of an amount of energy between identical, but *distinguishable* particles. The distribution of a fixed amount of energy among a number of identical particles depends upon the density of available energy states and the probability that a given state will be

occupied. The distribution function  $f(E)$  is the probability that a given particle is in energy state  $E$ . Three distinctly different distribution functions are found in nature:

Maxwell-Boltzman	Bose-Einstein	Fermi-Dirac
Identical but distinguishable particles	Identical indistinguishable particles with integer spin (bosons)	Identical indistinguishable particles with half-integer spin (fermions)
Example: ideal gas	Example: thermal radiation	Example: electrons in a metal

## Distinguishability of Particles

The Bose-Einstein and Fermi-Dirac distributions differ from the classical Maxwell-Boltzmann distribution because the particles they describe are indistinguishable. Particles are considered to be indistinguishable if their quantum wave packets overlap significantly. Two particles can be considered to be *distinguishable* if their separation is large compared to their wavelength. For example, the condition of distinguishability is met by molecules in gases under ordinary conditions. For air at room temperature, the molecules have a separation on the order of 3 nm<sup>1</sup> and wavelengths on the order of 0.03 nm, a factor of a hundred smaller. On the other hand, two electrons in the first shell of an atom are inherently indistinguishable because of the large overlap of their wavefunctions.

### “How Many Ways to Distribute 10 Units of Energy among 5 Distinguishable Particles?”

The answer to this question is precisely the same as the answer to how many distinct ways non-negative integers (“units of energy”, including zero) add up to 10, which is  $Q(5, 10) = 1001$ . This number is the number of *microstates* for the system, which in turn determines the energy distribution function. So, the  $Q$  function is an important function in Thermodynamics and in our daily lives (whether we knew it or not).

## A Computer Simulation

We have expended considerable cleverness in finding a closed mathematical formula for  $Q$ . In these days with computers that execute billions of instructions per second and where processing power is essentially free, a viable alternative is often a simple computer simulation of the situation. Run through all possible combinations of  $n$  non-negative integers, add them up, see what their sum is, count the number of times their sum equals the sum,  $s$ , that we were looking for, and there we have the result. This is essentially a recursive process: for each value (from zero up to and including  $s$ ) of the first integer (let’s call this at level 1), run through all values for the second integer (at level 2), and for each of these, run through all values for the third integer (level 3) and so on, until done. The computer simulation may provide you with data to guide you in the derivation of an analytical result. In any event, the computer simulation allows me to illustrate a recursive technique (all the above mumbo-jumbo was just a ponderous excuse for showing the following program, **MIADSUMR**).

The current value at level 1 is stored in VALUES(1), at level 2 in VALUES(2), and so on up to the total number of integers to add (NUMBERS). When we (somehow) have all the values defined, we compute the sum of all of them:

```
SUM-VALUES:
  CPYNV      SUM, 0;
  CPYNV      I, 1;
ADD-TO-SUM:
  CMPNV(B)   I, NUMBERS/HI(COUNT-IF-MATCH);
  ADDN(S)    SUM, VALUES(I);
  ADDN(SB)   I, 1/POS(ADD-TO-SUM);
```

When the sum is computed, we check to see if its value matches the sum we are looking for:

```
COUNT-IF-MATCH:
  CMPNV(B)   SUM, WANTED-SUM/NEQ(DONE);
  ADDN(SB)   COUNT, 1/POS(DONE);      /* if match: increment COUNT */
```

In any event, we end up at **DONE**. If this combination of VALUES added up to WANTED-SUM, the COUNT was increased. No rocket-science here. Note that we still don’t know how to keep track of the combinations.

<sup>1</sup> nm = nanometer, 10<sup>-9</sup> meter, 0.000 000 039 inches.

What we have shown above is the “bottom” of the recursion, where the real action takes place. Now, let’s look at the fluff part. Basically, the current level will be 1 greater than the previous level; at the current level we try successively all values from 0 to WANTED-SUM. (Remember that none of them can be greater than WANTED-SUM, if their sum has to *be* WANTED-SUM.):

```
[MIADSUMR]
ADDN      CUR-LEVEL, PREV-LEVEL, 1;: /* next level */
CMPNV(B)  CUR-LEVEL, NUMBERS/HI(SUM-VALUES);
CPYNV     VALUES(CUR-LEVEL), 0;
ADD-LOOP:
CMPNV(B)  VALUES(CUR-LEVEL), WANTED-SUM/HI(DONE);
CALLX     .MIADSUMR, MIADSUMR, *;
ADDN(SB)  VALUES(CUR-LEVEL), 1/POS(ADD-LOOP);
DONE:
```

Each time through the loop we call **MIADSUMR** again (*i.e.* recursively) to take care of all the higher levels. This is equivalent to starting the code over. When the program returns, control continues after the CALLX-statement. Study the above code; it is only six instructions (albeit with a higher than average percentage of loops, test, and exits).

## Programming Idioms

We have really not specified in any great detail how to generate the combinations and manage the process. This is not necessary, as the basic *structure* of the loops automatically leads to the desired pattern. What we are seeing here is an example of a programming *idiom*. This kind of structure is the one that is appropriate for situations where you are generating all combinations from a sequence.

```
for i1 = 0 to n do
  for i2 = 0 to n do
    for i3 = 0 to n do
      for i4 = 0 to n do
        for i5 = 0 to n do
          . . .
        end
      end
    end
  end
end
end
end
```

If we knew beforehand that there would be always 5 levels, we could write the code to generate the combinations as a series of nested for loops as shown on the left. What the recursive call does for us is to “telescope” all these loops into a single loop with a recursive call in the middle; a useful by-product of the telescoping is that the restriction to a fixed number of levels is relaxed.

The programming idiom is a very powerful concept. It is a formal (fancy) way of saying: “been there, done that”. It is often called a “pattern”, and a whole cult has grown up around pattern-oriented software architectures.

## Storage Attributes

Fundamental to the use of recursion is an understanding of three *storage attributes* that a variable can have:

- A variable with the *static* (STAT) attribute is shared (it is the same storage location) between all invocations (*i.e.* CALLXs) of the same program. Its value is unchanged from call to call until explicitly changed by the program.
- A variable with the *automatic* (AUTO) attribute is local (it is a different storage location) to each invocation of the program. Its value is undefined from call to call. Note that the variable is *not* automatically initialized to anything each time the program is called.
- A variable with the *parameter* (PARM) attribute is shared (it is the same storage location) between invocations of programs. Its value is “passed forth and back” from program to program.

An important issue connected to storage attributes is the problems of the *first time* (getting the process starting) and the *last time* (when the recursion bottoms out). The middle part is the easiest, you just keep calling the procedure. The first time around, there are values to be initialized and when returning to that level from the call of the procedure (and from all *its* calls), results to be printed out.

## Calling **MIADSUMR**

We would like to invoke (“call”) the program from the command line (or in a command) with:

```
CALL MIADSUMR PARM(5 10)
```

to find the number of different ways that (say) 5 values can up to (say) 10. Clearly, we must specify the parameters as follows to conform to the calling conventions of CL-programs:

```
DCL SPCPTR .PARM1 PARM;
DCL DD NUMBERS PKD(15,5) BAS(.PARM1); /* PKD(15,5): can come from command line */

DCL SPCPTR .PARM2 PARM;
DCL DD WANTED-SUM PKD(15,5) BAS(.PARM2);

DCL SPCPTR .PARM3 PARM;
DCL DD PREV-LEVEL BIN(4) BAS(.PARM3);

DCL OL PARMS(.PARM1, .PARM2, .PARM3) PARM EXT MIN(2);
```

We’ll explain the rôle of the third (optional) parameter shortly. We’ll use the number of parameters passed (MIN(2)) as a way of knowing that this is the first time, namely if that number is 2. Every time thereafter the number of parameters will be three as we are telling the next invocation what level we currently are on, so it can take of the next level (from its viewpoint our current level is *its* previous level, hence the name PREV-LEVEL). When **MIADSUMR** is called the first time, the third parameter does not exist, so when we call ourselves, we need to use a different operand list that includes a pointer (.ARG3) to the variable containing our CUR-LEVEL:

```
DCL SPCPTR .ARG3 AUTO;
DCL DD CUR-LEVEL BIN(4) AUTO;

DCL OL MIADSUMR (.PARM1, .PARM2, .ARG3) ARG;
DCL SYSPTR .MIADSUMR INIT("MIADSUMR", TYPE(PGM));


ENTRY * (PARMS) EXT;
SETSP .ARG3, CUR-LEVEL; /* point to our current level */
```

Note, that because the variables are declared to be AUTOMATIC, we cannot initialize the space pointer .ARG3 with the *static* INIT clause: SPCPTR .ARG3 INIT(CUR-LEVEL).

We wish to COUNT the number of times the sum matches what we want as well as the total number of combinations examined (COMBS), so the first time (when NBR-OF-PARMS is 2), we need to initialize these values to zero. Also, we need to initialize CUR-LEVEL to one:

```
DCL DD NBR-OF-PARMS BIN(2) AUTO;
DCL DD COUNT ZND(10,0) STAT;
DCL DD COMBS ZND(10,0) STAT;

...
STPLEN NBR-OF-PARMS;
CMPNV(B) NBR-OF-PARMS, 2/NEQ(THERE-AFTER);
FIRST-TIME:
CPYNV COUNT, 0;
CPYNV COMBS, 0;
CPYNV(B) CUR-LEVEL, 1/POS(=+2);
THERE-AFTER:
ADDN CUR-LEVEL, PREV-LEVEL, 1;:
```




Every time thereafter, CUR-LEVEL is PREV-LEVEL plus one. When CUR-LEVEL eventually becomes higher than NUMBERS the recursion bottoms out and it’s time to go add up the values to see if the sum is what we want:

```
CMPNV(B) CUR-LEVEL, NUMBERS/HI(SUM-VALUES);
```

Otherwise, we try all combinations of VALUES from 0 and up to WANTED-SUM, calling **MIADSUMR** to take care of all higher levels:

```
CPYNV VALUES(CUR-LEVEL), 0;
ADD-LOOP:
CMPNV(B) VALUES(CUR-LEVEL), WANTED-SUM/HI(DONE);
CALLX .MIADSUMR, MIADSUMR, *;
ADDN(SB) VALUES(CUR-LEVEL), 1/POS(ADD-LOOP);
DONE:
```



When we are DONE, we show the resulting values of COUNT and COMBS provided we are back at the first invocation (NBR-OF-PARMS = 2; alternatively - and perhaps clearer - we could have used CUR-LEVEL = 1):

```

      CMPNV(B)      NBR-OF-PARMS, 2/NEQ(RETURN);
      CPYBLAP      MSG-TEXT, COUNT, " ";
      CPYBLA      MSG-TEXT(16:10), COMBS;
      CALLI      SHOW-MESSAGE, *, .SHOW-MESSAGE;
RETURN:
      RTX          *;

```

For completeness, we show again the code to sum the values; note that the VALUES array (arbitrarily limited to 99 entries) must be STATIC to be shared by all invocations:

```

DCL DD VALUES(99) BIN(4) STAT;
DCL DD I      BIN(4);
DCL DD SUM    BIN(4);

SUM-VALUES:
  CMPNV(B)      NUMBERS, 0/EQ(DONE);
  ADDN(S)      COMBS, 1;
  CPYNV        SUM, 0;
  CPYNV        I, 1;
ADD-TO-SUM:
  CMPNV(B)      I, NUMBERS/HI(COUNT-IF-MATCH);
  ADDN(S)      SUM, VALUES(I);
  ADDN(SB)     I, 1/POS(ADD-TO-SUM);

COUNT-IF-MATCH:
  CMPNV(B)      SUM, WANTED-SUM/NEQ(DONE);
  ADDN(SB)     COUNT, 1/POS(DONE);

```

Alternatively, the static variables could all have been passed along as parameters, but that would have cluttered up the calling sequence. As usual, a choice must be made between explicitness and convenience. Too much use of hidden global (“behind the back”) variables is inherently bad, so a justification must be found to use this technique. *My* excuse is that I want to illustrate the various storage attributes.

## A Parameter That Is *Not* a Pointer

Because most high-level languages generate code that passes parameters via pointers we too have been using this standard convention, but, as we mentioned back in Chapter 5, we don’t *have* to. The calling sequence can directly use the level-parameters without going through a pointer to it. This actually simplifies matter a bit as we do not need to set a space pointer to the argument explicitly:

```

DCL SPCPTR .PARAM1 PARAM;
DCL DD NUMBERS PKD(15,5) BAS(.PARAM1);

DCL SPCPTR .PARAM2 PARAM;
DCL DD WANTED-SUM PKD(15,5) BAS(.PARAM2);

DCL DD PREV-LEVEL  BIN(4) PARAM;
DCL DD CUR-LEVEL   BIN(4) AUTO;

DCL OL      MIADSUMX (.PARAM1, .PARAM2, CUR-LEVEL) ARG;
DCL SYSPTR  .MIADSUMX INIT("MIADSUMX", TYPE(PGM));

DCL OL      PARMS(.PARAM1, .PARAM2, PREV-LEVEL) PARM EXT MIN(2);
ENTRY * (PARMS) EXT;
/* SETSP      .ARG3, CUR-LEVEL; not needed anymore */
STPLLEN     NBR-OF-PARMS;
...

```

An added advantage (☺) is that the recursive (“inner”) parts of the program cannot be called from most high-level languages. This is sometimes convenient.

## An Iterative Version, *MIADSUMI*

Guided by the structure of the recursive version it becomes clear that each time we perform a call to handle a higher level, the level number goes up by one; when we come back, the level number should be back to its original value. This observation leads us to the idea of simply counting the level number up and down as shown in the iterative version below; at least to my mind, without the recursive version as a guide, it is not immediately clear that the following code actually works:

```

DCL SPCPTR .PARM1 PARM;
DCL DD NUMBERS PKD(15,5) BAS(.PARM1);

DCL SPCPTR .PARM2 PARM;
DCL DD WANTED-SUM PKD(15,5) BAS(.PARM2);

DCL OL PARMS(.PARM1, .PARM2) PARM EXT MIN(2);

DCL DD COUNT      ZND(10,0);
DCL DD COMBS      ZND(10,0);
DCL DD VALUE (99) BIN(4);
DCL DD LEVEL      BIN(4);

ENTRY * (PARMS) EXT;
  CPYNV          COUNT, 0;
  CPYNV          COMBS, 0;
  CPYNV          LEVEL, 0;

FOR-EACH-LEVEL:
  ADDN(S)        LEVEL, 1;
  CMPNV(B)       LEVEL, NUMBERS/HI(SUM-VALUES);
  CPYNV          VALUE (LEVEL), -1;
CONTINUE:
  ADDN(S)        VALUE (LEVEL), 1;
  CMPNV(B)       VALUE (LEVEL), WANTED-SUM/NHI(FOR-EACH-LEVEL);
MORE:
  SUBN(SB)       LEVEL, 1/POS(CONTINUE);

DONE:
  CPYBLAP        MSG-TEXT, COUNT, " ";
  CPYBLA         MSG-TEXT(16:10), COMBS;
  CALLI         SHOW-MESSAGE, *, .SHOW-MESSAGE;
  RTX           *;

DCL DD I      BIN(4);
DCL DD SUM    BIN(4);

SUM-VALUES:
  CMPNV(B)    NUMBERS, 0/EQ(MORE);
  ADDN(S)     COMBS, 1;
  CPYNV      SUM, 0;
  CPYNV      I, 1;
ADD-TO-SUM:
  CMPNV(B)    I, NUMBERS/HI(COUNT-IF-MATCH);
  ADDN(S)     SUM, VALUE (I);
  ADDN(SB)    I, 1/POS(ADD-TO-SUM);

COUNT-IF-MATCH:
  CMPNV(B)    SUM, WANTED-SUM/NEQ(MORE);
  ADDN(SB)    COUNT, 1/POS(MORE);

```

A recursive version is often used as a natural step on the way to developing an iterative solution. It brings into focus the essence of the computation to be performed. We shall see that even more clearly in the Tower of Hanoi example.

### Sample Results (Combinatorial Explosion)

Here are the results for two sets of values; one where we ask how many different ways  $n$  values can add to  $k$ , holding  $k$  constant at 10 and varying  $n$  from 0 to 8; and a second run where we hold  $n$  constant at 6 and vary  $k$  from 0 to 8:

$n$	$k$	Count	Combinations
0	10	0	0
1	10	1	11
2	10	11	121
3	10	66	1,331
4	10	286	14,641
5	10	1,001	161,051
6	10	3,003	1,771,561
7	10	8,008	19,487,171
8	10	19,448	214,358,881

$n$	$k$	Count	Combinations
6	0	1	1
6	1	6	24
6	2	21	729
6	3	56	4,096
6	4	126	15,625
6	5	252	46,656
6	6	462	117,649
6	7	792	262,144
6	8	1,287	531,441



It is clear that the number of combinations grows too rapidly for exploring the results for values higher than about 10, thus prompting a search for a closed formula. In the next section we'll show a general technique for guessing such formulae.

### Method of Finite Differences

The method of finite differences can sometimes be used to guess a formula  $f(n)$  (but not to prove it). The method requires that the values of  $f(n)$  be given for integer values of  $n$  beginning with  $n = 0$  (rather than 1). We accomplish this by renaming  $n$  to  $n-1$ . To guess the formula for  $Q(n, k)$  we use the results of our second set of sample runs, arranging them in a row. Then we build a difference table beneath them by subtraction:

0		1	6	21	56	126	252	462	792	1287	2002	3003
	1	5	15	35	70	126	210	330	495	715	1001	
		4	10	20	35	56	84	120	165	220	286	
			6	10	15	21	28	36	45	55	66	
				4	5	6	7	8	9	10	11	
					1	1	1	1	1	1	1	
						0	0	0	0	0	0	

We continue generating rows by this process until it appears that we get a row of 0s. This does not always happen, but when it does, we can build a polynomial formula for  $f(n)$ . Using the first entries in each row, multiply the first entry of the first row by  $1/0!$ , the first entry of the second row by  $n/1!$ , the first entry of the third row by  $n(n-1)/2!$ , the first entry of the fourth row by  $n(n-1)(n-2)/3!$ , the first entry of the fifth row by  $n(n-1)(n-2)(n-3)/4!$ , etc., and then add the results. (Remember that  $0!$  is equal to 1.) We get:

$$f(n) = 0 \cdot 1/0! + 1 \cdot n/1! + 4 \cdot n(n-1)/2! + 6 \cdot n(n-1)(n-2)/3! + 4 \cdot n(n-1)(n-2)(n-3)/4! + 1 \cdot n(n-1)(n-2)(n-3)(n-4)/5! + 0$$

which simplifies to  $f(n) = (n^5 + 10n^4 + 35n^3 + 50n^2 + 24n)/120$ . After some fiddling around you may see that this is the same as  $f(n) = n(n+1)(n+2)(n+3)(n+4)/120$  (hint: divide successively by  $n, n+1, n+2, \dots$ ). Renaming  $n$  back to  $n+1$  (to undo the renaming of  $n$  to  $n-1$ ) we discover that we can write the function  $f(n)$  as  $(n+1)/1 \cdot (n+2)/2 \cdot (n+3)/3 \cdot (n+4)/4 \cdot (n+5)/5 = (n+k-1)!/n!(k-1)!$  for  $k = 6$ , which is just  $Q(n, k)$ . At this point we have *not* proved that this formula is correct, but at least we have a good guess for the formula, and we can try to prove it by other means, e.g., by induction.

### A Note About the Programs for This Chapter

Three programs for this chapter, **MIADSUMR**, **MIADSUMX**, and **MIADSUMI** are contained in the same text file (MIADSUMR). Each program ends with the PEND instruction to stop compilation of the remainder of the file. To compile MIADSUMX and MIADSUMI, you'll have to extract the programs from the common text file.

### The Tower of Hanoi

In a monastery in Hanoi there are three diamond rods. God placed 64 gold disks of different sizes - each with a central hole - on rod number 1 such that the largest disk is at the bottom of the stack and the smallest disk is on top. The monks' mission is to move all the disks onto rod number 2. Only one disk may be moved at a time and never may a larger disk be placed on a smaller one. If needed, disks may temporarily be placed on the third rod, again in such a manner that no larger disk rests on a smaller one.

This classical problem generalized to  $N$  disks can be solved by a recursive procedure that first moves all disks except the last one from the source rod to the spare rod, then moves the last - and largest - disk from the source rod to its final position on the destination rod, and finally moves all disks from the spare rod to the destination rod. Moving  $N-1$  disks is then performed in the same manner recursively until all disks have been moved. In ALGOL-like pseudo-code:

```

BEGIN INTEGER M;
  PROCEDURE MOVE (N, SOURCE, DEST, SPARE);
    INTEGER N, SOURCE, DEST, SPARE;
    BEGIN
      IF N > 1 THEN MOVE (N-1, SOURCE, SPARE, DEST);
      WRITE (OUT, "MOVE DISK ", N, " FROM ", SOURCE, " TO ", DEST);
      IF N > 1 THEN MOVE (N-1, SPARE, DEST, SOURCE)
    END;
    WRITE (OUT, "NBR OF DISKS = "); READ (IN, M);
    IF M > 0 THEN MOVE (M) SOURCE:(1) DEST:(2) SPARE:(3)
  END

```



When all disks have been moved the World will come to an end. Luckily, the total number of moves turns out to be the same as the number of bytes in the AS/400's address space ( $2^{64}$ ) so the monks have a long way to go.

It is not immediately clear how to write a non-recursive solution to the Tower of Hanoi puzzle. The recursive version is easy. Here is **MITWRHAN**:

```
DCL SPCPTR .PARM1 PARM;
DCL DD NBR-OF-DISKS PKD(15,5) BAS(.PARM1);

DCL DD SOURCE BIN(4) PARM;
DCL DD DEST  BIN(4) PARM;
DCL DD SPARE  BIN(4) PARM;

DCL OL PARMS(.PARM1, SOURCE, DEST, SPARE) PARM EXT MIN(1);

DCL SPCPTR .ARG1 AUTO;
DCL DD REMAINING-DISKS PKD(15,5) AUTO;
DCL DD ROD1  BIN(4) INIT(1);
DCL DD ROD2  BIN(4) INIT(2);
DCL DD ROD3  BIN(4) INIT(3);

DCL OL      MOVE-START(.ARG1, ROD1 , ROD2 , ROD3 ) ARG;
DCL OL      MOVE-OFF  (.ARG1, SOURCE, SPARE, DEST ) ARG;
DCL OL      MOVE-BACK (.ARG1, SPARE , DEST , SOURCE) ARG;
DCL SYSPTR .MITWRHAN INIT("MITWRHAN", TYPE(PGM));

DCL DD EDITED-NBR ZND(3,0);
DCL DD NBR-OF-PARMS BIN(2);
ENTRY * (PARMS) EXT;
  SETSPP      .ARG1, REMAINING-DISKS;
  STPLLEN     NBR-OF-PARMS;
  CMPNV(B)    NBR-OF-PARMS, 1/NEQ(THERE-AFTER);

FIRST-TIME:
  CPYNV       REMAINING-DISKS, NBR-OF-DISKS;
  CMPNV(B)    REMAINING-DISKS, 0/NHI(=+2);
  CALLX       .MITWRHAN, MOVE-START, *;;
  RTX        *;

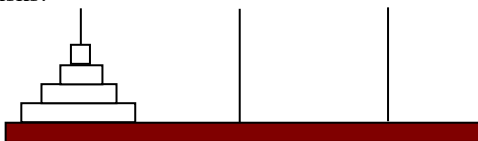
THERE-AFTER:
  CMPNV(B)    NBR-OF-DISKS, 1/NHI(=+3);
  SUBN       REMAINING-DISKS, NBR-OF-DISKS, 1;
  CALLX       .MITWRHAN, MOVE-OFF, *;;

  CPYBLAP    MSG-TEXT, "MOVE DISK", " ";
  CPYNV      EDITED-NBR, NBR-OF-DISKS;
  CPYBLA     MSG-TEXT(12:3), EDITED-NBR;
  CPYBLA     MSG-TEXT(17:5), "FROM";
  CPYNV      EDITED-NBR, SOURCE;
  CPYBLA     MSG-TEXT(24:3), EDITED-NBR;
  CPYBLA     MSG-TEXT(29:4), "TO";
  CPYNV      EDITED-NBR, DEST;
  CPYBLA     MSG-TEXT(35:3), EDITED-NBR;
  CALLI      SHOW-MESSAGE, *, .SHOW-MESSAGE;

  CMPNV(B)    NBR-OF-DISKS, 1/NHI(=+3);
  SUBN       REMAINING-DISKS, NBR-OF-DISKS, 1;
  CALLX       .MITWRHAN, MOVE-BACK, *;;
  RTX        *;
```

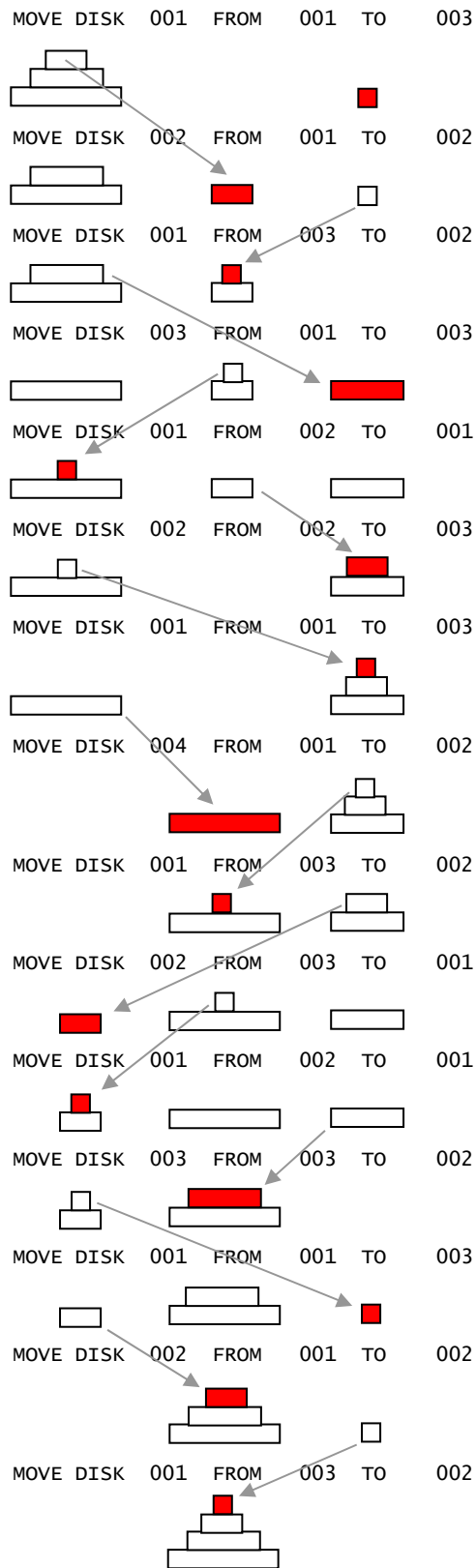
%INCLUDE SHOWMSG

The program produces instructions on how to move the disks. We'll try it on a smaller stack with only four disks:



Below is the result of

```
===> CALL MITWRHAN PARM(4)
```



Done! The number of moves with  $N$  disks is  $2^N - 1$ .

## Fibonacci Numbers

Sometimes simple recursion leads to exponential explosion. Consider the iterative solution to computing the famous Fibonacci series:

0 1 1 2 3 5 8 13 21 34 ...

where each term is the sum of the previous two:

```

MOVE 0 TO A
MOVE 1 TO B
ACCEPT N
PERFORM
    ADD B TO A
    SWAP (A, B)
    VARYING I FROM 1 BY 1
    UNTIL I > N

DISPLAY A

```

This is very efficient. The standard recursive solution is

$$F(n) = F(n-1) + F(n-2)$$

which is *very* much slower than the iterative version, because the number of calls *doubles* with each step. There is, however, a better recursive solution, that cuts that number down by a factor of two, thus avoiding the exponential growth:

compute  $m = n/2$  with remainder  $r$ , then

$$r = 0: F(n) = F(m+1)^2 + F(m)^2$$

$$r = 1: F(n) = F(m+1)^2 - F(m-1)^2$$

Here is an MI-implementation, **MIFIBNBR**, of this scheme:

```

DCL SPCPTR .PARM1 PARM;
DCL DD PARM-SEQNBR PKD(15,5) BAS(.PARM1);

DCL SPCPTR .PARM2 PARM;
DCL DD PARM-RESULT ZND(31,0) BAS(.PARM2);

DCL OL PARMS(.PARM1, .PARM2) PARM EXT MIN(1);

DCL SPCPTR .ARG1 AUTO;
DCL DD FIB-WANTED PKD(15,5) AUTO;

DCL SPCPTR .ARG2 AUTO;
DCL DD FIB-RESULT ZND(31,0) AUTO;

DCL OL MIFIBNBR(.ARG1, .ARG2) ARG;
DCL SYSPTR .MIFIBNBR INIT("MIFIBNBR", TYPE(PGM));

DCL DD TERM1 PKD(31,0) AUTO;
DCL DD TERM2 PKD(31,0) AUTO;

DCL DD N PKD(5,0) AUTO;
DCL DD M PKD(5,0) AUTO;
DCL DD R PKD(5,0) AUTO;

DCL DD NBR-OF-PARMS BIN(2);

ENTRY * (PARMS) EXT;
SETSPP .ARG1, FIB-WANTED;
SETSPP .ARG2, FIB-RESULT;
CPYNV N, PARM-SEQNBR;
CMPNV(B) N, 3/LO(STARTING-VALUES);

DIVREM M, N, 2, R; /* R = 0,1 FOR N EVEN,ODD */
ADDN FIB-WANTED, M, 1;
CALLX .MIFIBNBR, MIFIBNBR, *;
MULT TERM1, FIB-RESULT, FIB-RESULT;

CMPNV(B) R, 0/EQ(EVEN);
ODD:
CPYNV FIB-WANTED, M;

```

```

CALLX      .MIFIBNBR, MIFIBNBR, *;
MULT       TERM2, FIB-RESULT, FIB-RESULT;
ADDN(B)    FIB-RESULT, TERM1, TERM2/NNEG(DONE);

EVEN:
SUBN       FIB-WANTED, M, 1;
CALLX      .MIFIBNBR, MIFIBNBR, *;
MULT       TERM2, FIB-RESULT, FIB-RESULT;
SUBN(B)    FIB-RESULT, TERM1, TERM2/NNEG(DONE);

STARTING-VALUES:
CPYNV      FIB-RESULT, 1;
CMPNV(B)   N, 0/HI(DONE);
CPYNV      FIB-RESULT, 0;

DONE:
STPLEN     NBR-OF-PARMS;
CMPNV(B)   NBR-OF-PARMS, 1/EQ(SHOW-RESULT);
CPYNV(B)   PARM-RESULT, FIB-RESULT/NNAN(RETURN);

SHOW-RESULT:
CPYBLAP    MSG-TEXT, FIB-RESULT, " ";
CALLI      SHOW-MESSAGE, *, .SHOW-MESSAGE;;

RETURN:
RTX        *;

%INCLUDE SHOWMSG

```

Run it to get the 144<sup>th</sup> Fibonacci number:

```

==> CALL MIFIBNBR 144
From . . . : LSVALGAARD 02/15/01 18:29:55
0555565404224292694404015791808

```

Even better than a fast recursive method is, of course, a closed form if one exists. For the Fibonacci numbers there is one, namely:

$F(n) = \phi^n / \sqrt{5}$ , rounded to the *nearest* integer. Where  $\phi$  is the “golden ratio”  $1.61803\ 39887\ \dots = (1+\sqrt{5})/2$

That a closed form or a fast iterative method exists, does, of course, not matter as we are only showing the general technique.

## Chapter 29

### Changing Your Machine's Serial Number

#### ***What Legitimate Reason Can You Have For This?***

In two words: *Disaster Recovery*. Your company depends on a specific application. You faithfully backup your data every day. One day disaster strikes and your AS/400 is down for the day and the foreseeable future. Luckily you have arranged for a different system to be used for just such an emergency. Unfortunately, in his zeal to extract the maximum amount of money from you, the vendor from whom you bought the application requires that a “security code” be issued to you based on your machine’s serial number. Without the correct code, the application will not run. So, your application does not run on the backup machine! Getting a temporary code for the backup machine turns out to be difficult: it is 2am Saturday morning, the vendor is not too responsive (maybe not even in business anymore), you have not paid the maintenance fee, bureaucratic red tape is in the way, a million reasons and excuses. This is a time when you need to take control and change the serial number yourself.

#### **A True Horror Story**

The impetus for the present chapter is the following true incident. I quote the story literally:

“I’ve got a situation that’s starting to rear its REALLY ugly head. This situation occurred prior to my arrival here and the principal player in our organization with which this situation occurred is long gone. As you read this, please note that I AM NOT JOKING! Everything I present below is “factual” as has been told to me by IBM. We have two AS/400s (both 9406-510’s running V3R7) at two of our branches whose serial numbers begin with ‘44’. According to IBM, ‘44’ indicates that the box was manufactured outside the U.S. and a serial number beginning with ‘10’ indicates that the box was manufactured inside the U.S. The problem is that IBM’s records show these two systems as beginning with ‘10’ with the correct last 5 digits of the serial number. This apparently was not a problem with all of our previous dealings with IBM since I was told that, normally, IBM only refers to the last 5 digits of the serial number, ignoring the first 2 digits. Several weeks ago, with my new responsibilities to manage all of our AS/400s, I attempted to use SNDPTFORD to order SF98370 (PSP). The system dialed up IBM, connected, but was returned a message from the IBM connection that “Service requester not registered with IBM Service.” (CPF8C3B). After various calls to our marketing rep, the branch’s C.E., and several other IBM’ers patched through on conference calls, over *several weeks*, this is what I’m being told:

“Basically, IBM is, for reasons unknown to me or anyone at IBM that I’ve talked to, unable to change their systems to show that the serial numbers on the two affected boxes are, in fact, 44\* boxes, and not 10\*. What they \*are\* supposedly able to do is to “flip a bit” that stays ‘on’ for a period of time (I have not been told how long a period of time that is) that will allow me to use IBM to order cume PTFs and MF PTFs (we do not have a software contract with them, and I am unable to find paperwork that would tell me whether we have any with a BP from whom we may have obtained the system from. Man, this s\*cks!). Also, there is a unexplainable delay for when the “bit” is turned on; I was told I’d have to wait SIX weeks for the “bit” to be turned on!!! Upgrading the OS on these boxes supposedly wouldn’t solve the problem either, even with the subscription service, since IBM will still see our boxes as 44\* instead of 10\*. If I were to attempt an upgrade to V4Rwhatever, I would have to make absolutely certain that I had a wide enough window for the “bit” to remain turned on, or else I’d have to wait another SIX weeks to turn it back on.

“It gets better. Last week I politely complained to our marketing rep that this was causing undue hardship on our company and wanted someone, anyone, at IBM to fix this problem once and for all. It was then that she told me that we wouldn’t want to do that because if we do, all future software orders for these two boxes would have to be handled from overseas, which she strongly suggested that I would not want to do. Even if it could be done that way, she said, doing so would cause us to lose the ability to transfer our licenses over to any new AS/400 we would replace the existing AS/400s with (something that will probably happen within the next two years).

“Again, and in the words of Dave Barry, I am not making any of this up! This whole issue has the potential to explode to the point where my company discards the AS/400s with a more generic solution at the branches. In fact, I have already been called upon by my managers, as part of their new due diligence responsibility, to make the case for replacing the AS/400s we have with new iSeries boxes instead of a more generic server. I have made what I believe is a compelling case with them to stay on the iSeries architecture, but IBM certainly isn’t making my job any easier.”

Contumelious vendors are the main reason for this chapter.

## The **MATMATR** Instruction

We have already seen how to use the “Materialize Machine Attributes” instruction, **MATMATR**, to retrieve the internal clock value. Besides the clock value, a host of other information is available with **MATMATR**, in particular the Machine Serial Number. The second operand of **MATMATR** is a function code that tells the instruction which information you want materialized:

```
DCL SPCPTR .MCH-ATTR INIT(MCH-ATTR);
DCL DD MCH-ATTR CHAR(16) BDRY(16);
DCL DD MCH-BYTES-PROVIDED BIN(4) DEF(MCH-ATTR) POS(1) INIT(16);
DCL DD MCH-BYTES-RETURNED BIN(4) DEF(MCH-ATTR) POS(5);
DCL DD MCH-SERIAL-NUMBER CHAR(8) DEF(MCH-ATTR) POS(9);

MATATTR .MCH-ATTR, X'0004'; /* GET SERIAL NUMBER */
```

The serial number is returned in the right-most 7 bytes of the 8-character MCH-SERIAL-NUMBER field.

## Finding the SCV Function

The **MATMATR** instruction is executed as a Supervisor Call Vectored (SCV) function. See Chapter 18 and Appendix E for more information about SCVs. The **MATMATR** instruction has function number 83 according to Appendix E. The V4R4M0 address for this function is x'FFFFFFFFC4 F5AA98'. Using SST we can find LIC modules by address. Here is what we get after providing the function address:

```
Entry:
Address . . . . . : FFFFFFFFFC4 F5AA98
Name . . . . . : .#rmatma
Module Name . . . . : #rmatma
Nickname . . . . . : #RMMATMA
Text Address . . . . : FFFFFFFFFC4 F5AA98

F9=Display Text    F10=Display Data    F11=Display BSS
```

The module seems to be **#RMMATMA**; the name certainly looks promising. Looking at the SST display panel, it struck me that the function key prompts are actually rather revealing. A standard UNIX binary file consists of three *segments*: A **text** segment containing executable instructions, a **data** segment with global and static variables that have values known at compile time, and a **BSS** segment. The BSS segment gets its name from abbreviating the “Block Started by Symbol” pseudo-operation from the 1956-vintage IBM/704 assembler. The BSS segment holds information about the sizes of global and static variables that have not yet been initialized. *UNIX*? What has this to do with the AS/400?

## OS/400 is Developed Under UNIX

What we are seeing here is that IBM (Rochester) does *not* use the AS/400 for the development of OS/400. In fact, they use the Andrew (Carnegie-Mellon) Unix platform and toolkit these days. Some think that Rochester’s reluctance to use what they sell (“eat their own dogfood”) is a major contributor to some of the problems with the system. There is nothing new here though; the System/38 and CISC-based AS/400s software was developed on IBM mainframes under VM/CMS using tools not available “natively.” Rochester today is just continuing that (proud?) tradition. See the end of the Chapter for more on this.

## Contents of the Data Segment

Pressing F10 shows us the contents of the data segment. The addresses will be different on other releases, but the module name is likely to be the same. So, on a different release, you should be able to find the LIC module by name (**#RMMATMA**) rather than by address.

Here is the beginning of the data segment:

Address	FFFFFFFFC8	79AE60		
79AE60	00000000	16011000	00000000	16010000
79AE70	00000000	00000001	FE008001	18800000
79AE80	A0000000	A6004000	00008000	8000FFFF
79AE90	00040000	00000000	FFFFFFF	C4F5AC20
79AEA0	01000000	00000000	FFFFFFF	C4F5ACCC
79AEB0	01040001	00000000	FFFFFFF	C4F5AD68
79AEC0	01080101	00000000	FFFFFFF	C596A938
79AED0	01180000	00000000	FFFFFFF	C4F5B1F8
79AEE0	012C0100	00000000	FFFFFFF	C1585500
79AEF0	01300000	00000000	FFFFFFF	C4F5AE3C
79AF00	01340000	00000000	FFFFFFF	C4F5AF20
79AF10	01380000	00000000	FFFFFFF	C4F5AFAC
79AF20	013C0000	00000000	FFFFFFF	C4F5AFEC
79AF30	01400000	00000000	FFFFFFF	C4F5B0B0
79AF40	01440000	00000000	FFFFFFF	C4F5B108
79AF50	01480000	00000000	FFFFFFF	C4F5B160
79AF60	014C0000	00000000	FFFFFFF	C4F5B1B8
79AF70	01610000	00000000	FFFFFFF	C4F5B2A0

Serial number  
Machine clock  
Primary initial process template  
Machine init status record MISR  
UPS delay  
Vital product data  
Network attributes  
Date format  
Leap year adjustment  
Time powered on  
Timed power enable/disable  
Remote power enable/disable  
Auto power restart enable/disable  
Date separator  
Perform HW check on IPL

We here clearly have a table of function codes and corresponding addresses of service routines. I have indicated the name of each function as given in the MI reference manual.

## Get the Serial Number

The first entry found in the function table is the function (0004) to get the serial number. This function immediately calls entry .ioinfo\_get\_obs\_ser\_syscec in module IoHriPlmplolnfoSupport:

FFFFFFF C4:

```
F5AC20 7F6802A6 MFSPR 27 08 [". w]
F5AC24 7C7E1B78 OR 30 03 03 [". i]
F5AC28 39800010 ADDI 12 00 16 [". 0 .]
F5AC2C 919E0004 STW 12(30) 4 [". E .]
F5AC30 38600000 ADDI 03 00 0 [". - .]
F5AC34 61ABE9CC ORI 11 13 059852 ["/Zö] remember that R13 = 8000..0000,
F5AC38 4B008563 BLA FF008560 [". eA] hence R11=8000000000 00E9CC
branch with link
```

FFFFFFF FF:

```
008560 796B1764 RLDICR 11 11 02 61 [". , Å] R11 shift 2: 0000000000 03A730
008564 3D6BF458 ADDIS 11 11 -2984 [". 4i] + FFFFFFFF4 580000
008568 7D6903A6 MTSPR 09 11 [". N w]
00856C 4E800420 BCCTR 20 00 [". 0 .] branch to FFFFFFFF4 5BA730
```

FFFFFFF F4:

```
5BA730 3C40F402 ADDIS 02 00 -3070 [". 4 .] .ioinfo_get_obs_ser_syscec:
5BA734 FBA1FFE8 STD 29(01) -24 [". U~. Y]
5BA738 FBC1FFF3 STMD 30(01) -16 [". UA. 3]
5BA73C 7C0802A6 MFSPR 00 08 [". @. w]
5BA740 F8010028 STD 00(01) 40 [". 8. .]
5BA744 F821FF81 STDU 01(01) -128 [". 8. a]
5BA748 3C000010 ADDIS 00 00 16 [". . .]
5BA74C F8010008 STD 00(01) 8 [". 8. .]
5BA750 EBA2DB60 LD 29(02) -9376 [". 0su-]
5BA754 61AB4710 ORI 11 13 018192 ["/Zâ. ] R11=8000000000 004710
5BA758 4B005203 BLA FF005200 [". e. ]
```

FFFFFFF FF:

```
005200 796B1764 RLDICR 11 11 02 61 [". , Å] R11 shift 2: 0000000000 011C40
005204 3D6B8080 ADDIS 11 11 -32640 [". , 00] + FFFFFFFF80 800000
005208 7D6903A6 MTSPR 09 11 [". N w]
00520C 4E800420 BCCTR 20 00 [". 0 .] branch to FFFFFFFF80 811C40
```

FFFFFFF 80: .getSysHriObjPtr\_Fv

```
811C40 3C408024 ADDIS 02 00 -32732 [". 0 .] R2= FFFFFFFF80 240000
811C44 E86208E8 LD 03(02) 2280 [". YÄ. Y] R3=(FFFFFFF80 2408E8)
```

FFFFFFF 80:

```
2408D0 FFFFFFFF 80A51A00 FFFFFFFF 809D4B90
2408E0 FFFFFFFF F45B5AC0 FFFFFFFF 809D2118 ← goes into R3
2408F0 FFFFFFFF 82FB979E FFFFFFFF 82FB97BA
```

811C48 E8630000 LD 03(03) 0 [". YÄ ] R3=C00000BC6C 04C000, because:

FFFFFFF 809D2118: ioHriSysCec\_\_11ioHriSysCec in module IoHriStaticDataNuc  
Contains C00000BC 6C04C00

```
811C4C 4E800020 BCLR 20 00 [". 0 .] return(R3= addr(SRLNBR area))
```

The result of all this is that we obtain a very important address, namely the address to a (large) segment where the actual hardware is described. Before we delve into that, let's first show the contents of the segment at the address we have identified:

```

Address C00000BC6C 04C000
04C000 FFFFFFFF 80A51A40 00000000 00000001  .... 0v. ....
04C010 C3C5C393 88998940 40404040 40404040  CECI hri  ← Central Electronic
04C020 40404040 40404040 40404040 40404040  Compl ex
04C030 40404040 40404040 40404040 00000007  ....
04C040 00004040 40404040 40404040 40404040  ..
04C050 40000000 00000000 00000000 00000000  ....
04C060 40000000 00000000 00000000 00000000  ....
04C070 00000000 00000000 00000000 00000000  ....
04C080 00000000 00000065  C3C5C3F0 F1404040  .... ACEC01  ← Card Enclosure 01
04C090 40400000 00000000 00000000 00000000  ....
04C0A0 06D8E3C7 00FF0000 FFFFFFFF 8080DCD0  . QTG. .... 00u}
04C0B0 00000000 00000000 00000000 00000000  ....
04C0C0 00000000 00000000 00000000 00000000  ....
04C0D0 00000000 00000000 00000000 00000000  ....
04C0E0 00000000 00000000 00000000 00000000  ....
04C0F0 00000000 00000000 00000000 00000000  ....
04C100 00000000 00000000 00000000 00000000  ....
04C110 00000000 00000000 00000000 00000000  ....
04C120 00000000 00000000 00000000 00000000  ....
04C130 D3C8D9C9 F0F0F0F0 F0F0F000 00000000  LHRI 00000000. ....
04C140 00000000 00000000 00000000 00000001  ....
04C150 1F860438 062C0000 00000000 00000000  . f. ....
04C160 00000000 00000000 00000000 00000000  ....
04C170 C00000BC 6C04C178 FFFFFFFF 809CCD80  {..-%. Al.. 0æ00
04C180 00F1F0F5 F5F5F5D9 40400000 00000000  105555R  ← Serial number
04C190 00000000 00000000 FFFFFFFF 803EC5F0  .... 0. EO
04C1A0 C00000BC 6C041DF8 00000014 00000000  {..-%. 8. ....
04C1B0 C00000BC 6C041D80 00000007 00000000  {..-%. 0. ....
04C1C0 C00000BC 6C041D88 C00000BC 6C041DE8  {..-%. h{..-%. Y
04C1D0 04D8C781 00FFFFFF FFFFFFFF 803713F8  . QGa. .... 0. 8
04C1E0 80000000 00000000 00000000 00000000  0. ....
04C1F0 00000000 00000000 00000000 00000000  ....

```

If you change the value identified above as the serial number (e.g. with SST, or [better] using the method described in Chapter 7), you'll find that the value retrieved by the **MATMATR** (function 0004) instruction will be the changed value. [The value shown above is fake and does not refer to a real machine] When you save an object, the save file will contain the serial number of the machine on which you did the save (see Chapter 27 for more about save files). The serial number that is stored in the save file is the value found in the location described above. The "Retrieve Hardware Resource Information" API, **QGYRHRI**, also returns this serial number. For reasons to be clear in a moment, we'll call this piece of information the *real* serial number.

## The HDTA Object

We have learned that system objects start on a segment boundary, i.e. with offset zero. If we set the offset to zero for the address we discovered above and look at what is stored there, we see a signature of "HDTA", which we interpret to mean *Hardware Data*:

```

Address C00000BC6C 000000
000000 20AC0080 00000000 C00000BC 6C000000  . 0. 0. ... {..-%. ..
000010 00000000 00000000 00000000 00000000  ....
000020 00000000 00000000 C00000ED 51000000  HDTA. ... {.. 0é.  ← Hardware Data
000030 00000000 00000000 00000000 00000000  ....
000040 C00000BC 6C000048 FFFFFFFF FA0647C8  {..-%. 0. ... 3. àH
000050 00000000 00000000 00000000 00000000  ....
000060 00000000 00000000 00000000 00000000  ....
000070 00000000 00000001 00000000 000001F6  .... 6

```

Just browsing the segment reveals that all hardware items are described in it, e.g.:

```

Address C00000BC6C 005900
005900 00000000 00000000 00000000 00000000  ....
005910 00000000 00000000 00000000 00000000  ....
005920 00000000 00000000 00000000 00000000  ....
005930 C00000BC 6C009B80 C00000BC 6C00BF00  {..-%. 00{..-%. x.
005940 C00000BC 6C00CB00 D9858193 F1F060F5  {..-%. 0. Real 10-5  ← "Real " SN
005950 F5F5F5D9 4040F6F8 60F0F1C5 F5F5F5C1  555R 68-01E555A
005960 7F068B55 63D60000 00000030 0000002D  ". »f AO. ....
005970 00000003 01000000 00000000 00000000  ....

```



Many other interesting pieces of information appear in this segment: feature codes, model numbers, etc. This hardware information page is at x 'C00000BC6C 000000' on V4R5, V4R4, and all the way back to V3R7. Other releases may employ a different location. Follow the steps outlined in this Chapter on your system and tell us about your experience.

## System Values

Having changed the serial number to “106666R”, we displayed the **QSRLNBR** system value with this result:

```
====>DSPSYSVAL QSRLNBR

System value . . . . . : QSRLNBR
Description . . . . . : System serial number

Serial number . . . . . : 105555R
```

What is going on here? Why didn't the serial number seen by **DSPSYSVAL** change? The simple reason is that **DSPSYSVAL** takes its values from the **QWMSYSVAL** object (\*SVAL or type/subtype x'19D2') that only holds a *copy* of the real serial number. To complete the change we must change the copy as well. You can use SST to inspect/change the object:

```
Address 271E820E88 000000
000000 00010060 00828000 271E820E 88000000 . . . . b0 . . b. h. . .
000010 00010000 00000000 271E820E 88000100 }. . . . . b. h. . .
000020 800019D2 D8E6D4E2 E8E2E5C1 D3404040 0. . KQWMSYSVAL
000030 40404040 40404040 40404040 40404040
000040 40408000 0000BF00 00000060 FF1C0301 0. . x. . . . .
000050 7D447E7B 06EC0000 0DBF8EE4 BD000000 ' a=#. 0. . . xPU . . .
000060 00000000 00000000 3138C377 7D000000 . . . . . C' . . .

000100 40200084 005E8B00 9300005E 00008700 . . . d. ; ». l. . . ; . g.
```

The table structure is here that of a name (e.g. **QSRLNBR**) followed by an offset and a length. For **QSRLNBR** the offset is **0209** and the length is **0008**:

```
000760 D8D3C5C1 D7C1C4D1 404001C6 00024800 OLEAPADJ . F. . c.
000770 00000004 00000000 00000000 00000000 . . . . .
000780 D8E2D9D3 D5C2D940 40400209 00008400 QSRLNBR . . . d.
000790 04000001 00000000 00000000 00000000 . . . . .
0007A0 D8C1C3C7 D3E5D340 40400211 00500500 QACGLVL . . . &.
0007B0 04000005 00000000 00000000 00000000 . . . . .
```

The complete offset calculation proceeds now as follows:

- 1) the offset to the associated space is **000100**
- 2) add to this the *base offset* found in the first two bytes of the associated space, namely **4020** to get 4120
- 3) add to this the offset value found in the table, namely 0209 to get 4329 (all hex, of course)
- 4) at that place you find the table entry value, namely “ 105555R”:

```
0042B0 40400002 D8C3E3D3 40404040 4040D8E2 . . . QCTL QS
0042C0 E8E24040 40404040 D8E2E8E2 D6D7D940 YS QSYSOPR
0042D0 4040D8E2 E8E24040 40404040 5CD5D6D5 QSYS *NON
0042E0 C5404040 40400000 00000000 00000000 E . . . . .
0042F0 61D4C4E8 0002FFFF C5D5E4E4 E2405BF0 /MDY. . . . ENUUS $0
004300 F1F1F0F0 F0F0F0F0 F0F0F0F0 F0F0F0F0 1100000000000000
004310 F0F0F0F0 F0F0F0F0 F0F0F0F0 F0F05CC4 00000000000000*D
004320 D4D7E4E2 D9D1D6C2 F040F1F0 F5F5F5F5 MPUSRJOB 105555
004330 D95CD5D6 D5C54040 40404040 40404040 R*NONE
004340 40404040 40404040 40404040 40404040
004350 40404040 40404040 40404040 40404040
```

Note the leading space. Finally change the value to suit.

## What Happens at the Next IPL?

As we would expect (and hope), an IPL resets both serial numbers back to their “Factory Installed” values. This is good, as we did not wish to change the serial number permanently; only to recover from specific disaster-related conditions.

## Place of Manufacture

Serial numbers starting with 10 were all manufactured in Rochester. Serial numbers starting with 78 were manufactured in Mexico (AS/400's are no longer manufactured there), 44 in Italy and 65 in Ireland.

## The "Software" Serial Number

At times IBM have refused to add software to a customers order record without a so-called "software serial number". There actually *is* a software serial number for each system. This serial number seems to be part of the administrative procedure needed to obtain a "system password" that will allow you to start a machine that has been powered off for an extended time (like when being sold and transferred to another owner).

### Pure conjecture (Probably Incorrect):

I believe that the software serial number can be found as follows. Scan for the text string "**SmLdl Object**" within the HDTA area. Several lines further down you'll see a 9-digit number with a hyphen: **10-8120421**. This is the software serial number as far as I can tell:

Address	C00000BC6C	00C900			
00C900	FFFFFFF	80E5FF60	FFFFFFF	80E5FF28	.... 0V. .... 0V. .
00C910	E294D384	8940D682	918583A3	E5F4D9F1	<b>SmLdl Object</b> V4R1
00C920	D9858193	F1F060F5	F5F5F5D4	4040F6F8	Real 10-5555M 68
00C930	60C2F0C3	F2F14040	<b>82656AC4</b>	<b>8C360000</b>	-B0C21 bA;D0. .
00C940	00030000	00000000	00000001	00000000	.....
00C950	00000000	00000000	00000000	00000000	.....
00C960	00000000	00000000	00000000	00000000	.....
00C970	0000F1F0	60F8F1F2	F0F4F2F1	00000000	<b>10-8120421</b> .....
00C980	00000000	00000000	00000000	00000000	.....
00C990	E294C386	87E2A392	001F0002	00000000	SmCfGStk.....
00C9A0	<b>7FB7751A</b>	34E38001	C1848440	4040F400	"%I. . T0. Add 4.
00C9B0	<b>82658B8F</b>	0CE88001	C1848440	4040F300	bA]±. Y0. Add 3.
00C9C0	00000000	00000000	00000000	00000000	.....

There is also a **timestamp** of installation of the software.

## A Little More History

A reader contributed the following observations:

The compilers originally developed for the S/38 and CPF were all developed in IBM Rochester. With the beginning of the AS/400 development (Silverlake) project, IBM's corporate direction had shifted, and all compiler related development for all IBM platforms was moved to TOROLABS in Toronto, Canada.

In the timeframe of 1990 to 1994, during the development of ILE and the CISC to RISC migration strategy and the W-Code, IBM Toronto Labs made modifications to the old OPM/EPM C/400 compiler to allow "binding" (APTA, the so-called Application Performance Tuning Aid), which allowed binding several smaller \*PGM objects into one larger consolidated \*PGM object. This was used internally to develop the "next generation" ILE C/400 compiler and runtime support.

Meanwhile, IBM Rochester came up with their own "dialect" of W-Code, called "New MI" (aka. NMI), which looks a lot like W-Code, but is simplified somewhat in the actual binary format. Most, if not all, of the "OX" components that translate from NMI to native RISC PowerPC code are written in Modula-2, which was in favor at that time, prior to the adoption of C++ for the SLIC kernel development project.

Note: if a product was written in PL/MI, then it was done under VM/CMS, as apparently no one ever bothered to "port" the PL/MI compiler from VM/CMS to OS/400. The output of PL/MI is not MI assembler source, but rather, an actual "binary" MI template, in 80-byte card image format, which can be "punched" via the virtual card punch under VM/CMS, and then transported to an actual AS/400 machine, (probably via SNADS). It can then be read in and processed (encapsulated) into a \*PGM object, using tools or an API like **QSCCRTPG**. Similarly, parts of the "core" of CISC-based OS/400 are written in PL/MP, which also ran under VM/CMS. The resulting IMPI instruction stream objects were similarly "ported" to the OS/400 (probably via a similar mechanism). I am not certain of where the Modula-2 compiler ran (what platforms).

Now, for the RISC PowerPC machines, the C++ compiler used for the SLIC kernel development runs under AIX on RS/6000s. With the V5R1 announcement of a "new" ILE C/400 compiler "base" that runs under PASE, it is possible that we may actually see a "shift" here, and it may become possible for IBM to begin to use OS/400 to develop more parts of OS/400 in the future. (Whether they will actually do this or not remains to be seen.).

Blank

## Chapter 30

# The Advanced Encryption Standard

### *The Need for a New Encryption Standard*

When the DES encryption standard was adopted in 1977 it was stipulated that it be reviewed every five years. By 1987 it was already apparent that a replacement for DES was needed. It nevertheless took another fourteen years before a replacement was adopted (actually, final adoption is still pending at the time (2001) of writing). The then approving body, the National Bureau of Standards (NBS, now the National Institute of Standards and Technology, NIST) requested the National Security Agency's (NSA) help in evaluating the DES algorithm submitted by IBM. The NSA succeeded in weakening the key length from 128 bits to 56 bits. The NBS published this NSA modified DES algorithm. This was very likely the result of a misunderstanding between the NSA and the NBS. The NSA thought that DES was to be hardware only. The NSA has later said that DES was one of its greatest mistakes. Had they known that the details would be released to the public, they would never have agreed to it. Luckily, the mood has changed in the intervening years with most people agreeing that security by obscurity is not the way to go.

### *The New Encryption Standard*

In 1997 NIST invited the public to submit proposals for a new ("Advanced") encryption standard (AES). In 1998, NIST announced the acceptance of fifteen candidate algorithms and requested the assistance of the cryptographic research community in analyzing these candidates. The resulting analysis included an initial examination of the security and efficiency characteristics for each algorithm. NIST reviewed the results of this preliminary research and selected MARS, RC6™, Rijndael, Serpent and Twofish as finalists. Having reviewed further public analysis of the finalists, NIST has decided to propose **Rijndael** as the Advanced Encryption Standard (AES). The research results and rationale for this selection are documented at this URL: <http://csrc.nist.gov/encryption/aes/round2/r2report.pdf>. The current status of the AES can be found at: <http://csrc.nist.gov/encryption/aes/rijndael/>. The winning entry, *Rijndael*, was submitted by two Belgian cryptographers: Joan Daemen and Vincent Rijmen. For more information about the authors, you can visit their home page at <http://www.esat.kuleuven.ac.be/~rijmen/rijndael/>. It seems significant that the winning entry is of non-US origin and is freely available overseas (nevertheless, the standards document still talks about export restrictions!).

The Rijndael entry is of considerable simplicity and mathematical elegance. It only uses "constant-time" operations like table-lookup and exclusive or. The algorithm works on 16-byte blocks, with a key length that can be varied from 128 bits to 256 bits. Only 128-bit keys will be covered in our treatment of the AES.

### *The Rounds*

A cipher *round*-function is performed repeatedly on the so-called *internal state* of the algorithm. The state (as the input and output block size) is 128 bits long and consists of four 32-bit words. Initially, the input block is copied to the state; then the round-function is performed on the state a certain number of times (10 times for a 128-bit key), and finally, the resulting state is copied to the output block.. From the key value a set of sub-keys is built, one for each round. These sub-keys are referred to as a *key schedule*. During a round, the input bytes are combined with bytes of the key schedule through a mixture of substitution and carry-less addition (exclusive or). Substitution is performed as table-lookup in a large *S-box*. The S-box is simply a "scrambler", but is constructed carefully so that it has no "fixed-points", *i.e.* where  $S(a) = a$ , as well as several other desirable mathematical properties. The exact contents of the S-box are not really important to the Rijndael algorithm. You can even substitute the contents of the S-box with your own values if you fear that the "official" version may contain hidden "trap-doors". In addition to the S-box there are also a number of permutations of the state and the input bytes. It is possible to combine all these operations with the S-box to form one single (but larger) "box". The combined box is a large table. Using it improves performance on machines that can process more than 8 bits in parallel (most machines including the AS/400 fall in this category).

For performance reasons the state is actually *two* times 128 bits long. In the code that follows, the state is represented as eight 32-bit quantities: s0, s1, s2, s3, and t0, t1, t2, t3. During one round s0, s1, s2, and s3 are used as input values, while t0, t1, t2, and t3 are used as output values. In the next round this is reversed. The code for a typical round looks like this (expressed in C-like notation):

```
t0 = Te0[s0>>24] ^ Te1[(s1>>16) & 0xff] ^ Te2[(s2>> 8) & 0xff] ^ Te3[s3 & 0xff] ^ rk[4];
t1 = Te0[s1>>24] ^ Te1[(s2>>16) & 0xff] ^ Te2[(s3>> 8) & 0xff] ^ Te3[s0 & 0xff] ^ rk[5];
t2 = Te0[s2>>24] ^ Te1[(s3>>16) & 0xff] ^ Te2[(s0>> 8) & 0xff] ^ Te3[s1 & 0xff] ^ rk[6];
t3 = Te0[s3>>24] ^ Te1[(s0>>16) & 0xff] ^ Te2[(s1>> 8) & 0xff] ^ Te3[s2 & 0xff] ^ rk[7];
```

Let's take one of these statements apart:

t0 = Te0[s0>>24]	Use first byte of s0 as subscript into Te0 and get that word,
^ Te1[(s1>>16) & 0xff]	XOR result with Te1 subscripted with second byte of s1,
^ Te2[(s2>> 8) & 0xff]	XOR result with Te2 subscripted with third byte of s2,
^ Te3[s3 & 0xff]	XOR result with Te3 subscripted with last byte of s3,
^ rk[4];	XOR result with a key from the key schedule

The tables Te0, Te1, etc, hold the combined values of the permutation and S-box tables and look like this:

```
DCL DD TE0(0:255) CHAR(4) BDRY(8) INIT(
X'C6363A5', X'F87C7C84', X'EE777799', X'F67B7B8D',
X'FFF2F20D', X'D66B6BBD', X'DE6F6FB1', X'91C5C554',
DCL DD TE1(0:255) CHAR(4) BDRY(8) INIT(
X'A5C6363', X'84F87C7C', X'99EE7777', X'8DF67B7B',
X'0DFFF2F2', X'BDD66B6B', X'B1DE6F6F', X'5491C5C5',
DCL DD TE2(0:255) CHAR(4) BDRY(8) INIT(
X'63A5C63', X'7C84F87C', X'7799EE77', X'7B8DF67B',
X'F20DFF2', X'6BBD66B', X'6FB1DE6F', X'C55491C5',
DCL DD TE3(0:255) CHAR(4) BDRY(8) INIT(
X'6363A5C6', X'7C7C84F8', X'777799EE', X'7B7B8DF6',
X'F2F20DFF', X'6B6BBD6', X'6F6FB1DE', X'C5C55491',
```

Note that the other tables, Te1, etc, are just the basic table Te0 with each entry rotated 8 bits progressively to the right. Having the “rotate” be *pre-computed* speeds processing.

The whole algorithm is actually nothing but a very long series of such manipulations. The result of each operation is forming the input to the next. Because the ‘exclusive or’ operations are associative, we may perform them in any order that is convenient. An MI-implementation of the above code fragment would be:

```
CPYBTL      B, S0, 0, 8; XOR      T0, TE0(B), RK( 4);
CPYBTL      B, S1, 8, 8; XOR(S)   T0, TE1(B);
CPYBTL      B, S2, 16, 8; XOR(S)  T0, TE2(B);
CPYBTL      B, S3, 24, 8; XOR(S)  T0, TE3(B);
```

where:

```
DCL DD S0 CHAR(4); DCL DD S1 CHAR(4); DCL DD S2 CHAR(4); DCL DD S3 CHAR(4);
DCL DD T0 CHAR(4); DCL DD T1 CHAR(4); DCL DD T2 CHAR(4); DCL DD T3 CHAR(4);

DCL DD B BIN(4) UNSGND;
DCL DD RK(0:43) CHAR(4) BDRY(8); /* the key schedule */
```

## Copy Bits Logical (CPYBTL)

The “Copy Bits Logical” operation, **CPYBTL**, copies the unsigned bit string *source* operand starting at the specified *offset* for a specified *length* to the *receiver*:

CPYBTL            *Receiver, Source, Offset, Length*

The length of the receiver cannot exceed four bytes and the length operand must be an immediate value between 1 and 32. The selected bits are treated as an unsigned bit string and copied right adjusted to the receiver and padded on the *left* with binary zeroes. Just what we need to be able to extract single bytes from the 32-bit state variables, to then be used as subscript values.

## Generated RISC Code for MI-Version

Let's look at the generated code for the following typical line:

```
CPYBTL      B, S2, 16, 8; XOR(S)  T0, TE2(B);
```

The CPYBTL instruction is very efficient, it basically just directly loads the byte we want:

```
001518 88AC004A LBZ      05(12)      74 [hð ¢] ; pick up third byte of S2
00151C 90AC0060 STW      05(12)      96 [°ð -] ; store it as word in variable B
```

No shifting, masking, or other extraction is needed. Continuing:

```
001520 380C0918 ADDI     00 12      2328 [...] ; load address of TE2 table
001524 808C0060 LWZ      04(12)      96 [øð -] ; load B (again)
```

The optimizer didn't do a good job here as there is no reason to reload B (it was already in a register).

```
001528 80AC0050 LWZ      05(12)      80 [øð &] ; load T0
00152C 5484103A RLWINM   04 04 02 00 29 [èd··] ; multiply B by 4 to index a word
001530 7C802214 ADD      04 00 04      [ø···] ; add TE2 base address
001534 7FC40088 TD       30 04 00      [°D h] ; check subscript
001538 80040000 LWZ      00(04)      0 [ø·] ; load TE2[B]
00153C 7C042A78 XOR      04 00 05      [ø··î] ; XOR with T0 already loaded
001540 908C0050 STW      04(12)      80 [°ð &] ; store T0
```

Again, the optimizer doesn't remove the superfluous load and store of T0, plus the code insists on checking that the subscript is valid. In total, that slows the process by at least a factor of two.

## Generated RISC Code for C-Version

Since the AES code is available in C, we can compile the C code for:

```
t0 = Te0[s0>>24] ^ Te1[(s1>>16) & 0xff] ^ Te2[(s2>> 8) & 0xff] ^ Te3[s3 & 0xff] ^ rk[4];
```

Here is the generated code for this fragment from the C-module:

```
005420 815FFFE0 LWZ      10(31)      -20 [a·.ö]
005424 79460620 RLDICL   06 10 00 56 [ä···] s3 & 0xff
005428 78C81764 RLDICR   08 06 02 61 [îH·À]
00542C 79080020 RLDICL   08 08 00 32 [····]
005430 39240C30 ADDI     09 04      3120 [...]
005434 7CA94214 ADD      05 09 08      [øzâ·]
005438 7FC54888 TD       30 05 09      [°Eçh]
00543C 81850000 LWZ      12(05)      0 [ae··] Te3[] to R12

005440 815FFFE8 LWZ      10(31)      -24 [a·.Y]
005444 5546C23E RLWINM   06 10 24 08 31 [iäB·] (s2>> 8) & 0xff
005448 78C80620 RLDICL   08 06 00 56 [îH··]
00544C 79051764 RLDICR   05 08 02 61 [···À]
005450 78A50020 RLDICL   05 05 00 32 [iv··]
005454 39440830 ADDI     10 04      2096 [·ä··]
005458 7CCA2A14 ADD      06 10 05      [ø···]
00545C 7FC65088 TD       30 06 10      [°F&h]
005460 81060000 LWZ      08(06)      0 [a···] Te2[] to R8

005464 80BFFFE4 LWZ      05(31)      -28 [øx·U]
005468 54A6843E RLWINM   06 05 16 16 31 [èwd·] (s1>>16) & 0xff
00546C 78C50620 RLDICL   05 06 00 56 [îE··]
005470 78A61764 RLDICR   06 05 02 61 [îw·À]
005474 78C60020 RLDICL   06 06 00 32 [îF··]
005478 38A40430 ADDI     05 04      1072 [·u··]
00547C 7FC53214 ADD      30 05 06      [°E··]
005480 7FDE2888 TD       30 30 05      [°ú·h]
005484 80DE0000 LWZ      06(30)      0 [øú··] Te1[] to R6

005488 83DFFFE0 LWZ      30(31)      -32 [cÿ·\]
00548C 57DE463E RLWINM   30 30 08 24 31 [iüä·] (s0>>24) & 0xff
005490 7BDE1764 RLDICR   30 30 02 61 [°ü·À]
005494 7BDE0020 RLDICL   30 30 00 32 [°ü··]
005498 3BA40030 ADDI     29 04      48 [·u··]
00549C 7F9DF214 ADD      28 29 30      [°·2·]
0054A0 7FDCE888 TD       30 28 29      [°üyh]
0054A4 83DC0000 LWZ      30(28)      0 [cü··] Te0[] to R30

0054A8 7FC63278 XOR      06 30 06      [°F·î] R6 = R30 xor R6
```

```

0054AC 7CC84278 XOR      08 06 08      [0x00000000] R8 = R8 xor R6
0054B0 7D066278 XOR      06 08 12      [0x00000000] R6 = R8 xor R12
0054B4 7CCC3A78 XOR      12 06 07      [0x00000000] R12= R6 xor rk[4]
0054B8 919FFFF0 STW      12(31)      -16 [0x00000000] t0 = result

```

Although the operations are executed in a different order than for the MI-code, the number of instructions per assignment to a state variable is about the same (10 and 11). *C* was actually designed to be good at this kind of bit-diddling and predictably performs well.

## Optimized RISC Code for C-Version

The **CHGPGM** command has an optimize option:

```
====> CHGPGM PGM(MYPGM) OPTIMIZE(*FULL)
```

Setting the value to *\*FULL* invokes a very aggressive optimizer. Unfortunately, IBM has decided not to make that optimizing level available for OPM-programs. Let's see how well it does on our *C*-program. The optimizer works on a *\*PGM* object, not on the *\*MODULE* object, so we must first bind the *C*-program.

The optimizer does an *outstanding* job. Here is the result of decompiling the same line of code as above (I have rearranged the lines to bring together things that belong together):

```

00009C 571ED5BA      RLWINM 30,24,26,22,29
000068 38872038      ADDI  4,7,8248
0000A4 7F84F02E      LWZX  28,4,30
0000B4 541B55BA      RLWINM 27,0,10,22,29
00004C 39071C38      ADDI  8,7,7224
0000B8 7F48D82E      LWZX  26,8,27
0000BC 559995BA      RLWINM 25,12,18,22,29 ; (s2>>8) & 0xff
000044 38C71838      ADDI  6,7,6200         ; get base of Te2
0000C0 7F26C82E      LWZX  25,6,25         ; Te2[]
0000C4 7F5BCA78      XOR   27,26,25        ; XOR into t0
0000C8 555A15BA      RLWINM 26,10,2,22,29
000074 38A72438      ADDI  5,7,9272
0000CC 7F25D02E      LWZX  25,5,26
0000D0 7F9EDA78      XOR   30,28,27
0000D8 7F3CF278      XOR   28,25,30
0000D4 83690010      LWZ   27,0x10(9)
0000DC 7F7EE278      XOR   30,27,28

```

Only four instructions are needed to extract the byte value, lookup its substitution value in the table, and then XOR'ing the result into the state. It gets even better further down because the table base addresses are already loaded into registers. We are now down to *three* instructions per line. *This cannot be improved upon:*

```

000050 570B95BA      RLWINM 11,24,18,22,29
000054 7F66582E      LWZX  27,6,11
000058 559C55BA      RLWINM 28,12,10,22,29 ; (s2>>16) & 0xff
00005C 7C88E02E      LWZX  4,8,28          ; Te1[]
000060 7C9ADA78      XOR   26,4,27         ; XOR into t1
000080 555CD5BA      RLWINM 28,10,26,22,29
000084 7F64E02E      LWZX  27,4,28
00008C 541E15BA      RLWINM 30,0,2,22,29
000090 7FA5F02E      LWZX  29,5,30
000094 7F7CD278      XOR   28,27,26
000098 83490014      LWZ   26,0x14(9)
0000A0 7FBBE278      XOR   27,29,28
0000A8 7F5DDA78      XOR   29,26,27

```

The conclusion is that *for this type of problem*, the combination of *C* and *full* optimization cannot be beat. We shall therefore present the full AES algorithm in *C* as well (**AES**). The MI-version is available as **MIAES**.

## C-Version of AES

A problem with the *C*-code is that you have to express extraction of bytes as shifts and logical ANDs in a roundabout way that partly obscures the simplicity of the manipulations. A good test of the readability of a program is the *telephone* test: can you read the code out loud over the telephone to someone having to type it in at the other end? The official AES *C*-code has a hard time passing that test. This phenomenon is sometimes called "code clutter". The standard solution to this problem is to use the *C*-preprocessor to



encapsulate the ugly code. The preprocessor is often misused to create a private “mini”-language or to make *C* look like Pascal or worse. In recognition of this, the designers of Java deliberately excluded a Java pre-processor from the language. But there are cases where the pre-processor *can* be used to make the program clearer, simply by making it shorter, and the AES algorithm is one of them.

## Defining an Encryption Round

Starting from the calculation of the *t0* part of the state:

```
t0 = Te0[s0>>24] ^ Te1[(s1>>16) & 0xff] ^ Te2[(s2>> 8) & 0xff] ^ Te3[s3 & 0xff] ^ rk[4];
```

We can “paraphrase” it as a specific case of the following *template* or definition:

```
#define ROUND_E(r, out, in0, in1, in2, in3, key) \
    out = Te0[(in0 >> 24)] ^ \
          Te1[(in1 >> 16) & 0xff] ^ \
          Te2[(in2 >> 8) & 0xff] ^ \
          Te3[(in3 >> 0) & 0xff] ^ key
```

namely:

```
ROUND_E(1, t0, s0, s1, s2, s3, rk[4]);
```

You may ask: what does the “**r**” do? The answer is: nothing, but it gives the reader a handy way of keeping track of which round is being performed by using `ROUND_E` successively with **r** being 1, 2, 3, ...

With this abbreviated way of expressing what goes into the state and what comes out, we can write the encryption routine as shown below. First some preliminary notation: the text we are about to encrypt is called the *plaintext* (*pt*), the result of the encryption is the *ciphertext* (*ct*), the key schedule is stored in the *round key* array (*rk*). A special routine (see below) is needed to expand the *encryption key* into the key schedule.

Here then is the code (you can easily see the regularities in how the source bytes are shifted from sub-round to sub-round - it takes four 32-bit operations to complete a 128-bit round):

```
void AES_Encrypt(const u32 rk[], const u32 pt[], u32 ct[]) {
    u32 s0, s1, s2, s3, t0, t1, t2, t3;

    s0 = pt[0] ^ rk[0];
    s1 = pt[1] ^ rk[1];
    s2 = pt[2] ^ rk[2];
    s3 = pt[3] ^ rk[3];

    ROUND_E(1, t0, s0, s1, s2, s3, rk[4]);
    ROUND_E(1, t1, s1, s2, s3, s0, rk[5]);
    ROUND_E(1, t2, s2, s3, s0, s1, rk[6]);
    ROUND_E(1, t3, s3, s0, s1, s2, rk[7]);

    ROUND_E(2, s0, t0, t1, t2, t3, rk[8]);
    ROUND_E(2, s1, t1, t2, t3, t0, rk[9]);
    ROUND_E(2, s2, t2, t3, t0, t1, rk[10]);
    ROUND_E(2, s3, t3, t0, t1, t2, rk[11]);

    ...

    ROUND_E(8, s0, t0, t1, t2, t3, rk[32]);
    ROUND_E(8, s1, t1, t2, t3, t0, rk[33]);
    ROUND_E(8, s2, t2, t3, t0, t1, rk[34]);
    ROUND_E(8, s3, t3, t0, t1, t2, rk[35]);

    ROUND_E(9, t0, s0, s1, s2, s3, rk[36]);
    ROUND_E(9, t1, s1, s2, s3, s0, rk[37]);
    ROUND_E(9, t2, s2, s3, s0, s1, rk[38]);
    ROUND_E(9, t3, s3, s0, s1, s2, rk[39]);

    LAST_E(10, ct[0], t0, t1, t2, t3, rk[40]);
    LAST_E(10, ct[1], t1, t2, t3, t0, rk[41]);
    LAST_E(10, ct[2], t2, t3, t0, t1, rk[42]);
    LAST_E(10, ct[3], t3, t0, t1, t2, rk[43]);
}
```

We have omitted the middle rounds, but the pattern is clear nevertheless. The **AES** program gives the complete code. Note how the output of the rounds alters between the “s” state variables and the “t” state variables and how these are used as input to the following round.

The last round is somewhat special because it uses a separate table (Te4):

```
#define LAST_E(r, out, in0, in1, in2, in3, key) \
    out = (Te4[(in0 >> 24) & 0xff] & 0xff000000) ^ \
          (Te4[(in1 >> 16) & 0xff] & 0x00ff0000) ^ \
          (Te4[(in2 >> 8) & 0xff] & 0x0000ff00) ^ \
          (Te4[(in3 >> 0) & 0xff] & 0x000000ff) ^ key \

static const u32 Te4[256] = {
    0x63636363U, 0x7C7C7C7CU, 0x77777777U, 0x7B7B7B7BU,
    0xF2F2F2F2U, 0x6B6B6B6BU, 0x6F6F6F6FU, 0xC5C5C5C5U,
    0x30303030U, 0x01010101U, 0x67676767U, 0x2B2B2B2BU,
    ...
}
```

Note how the plaintext (XORed with the round key portions) was used as the initial state (s0, ...) and how the last round stores the state (t0, ...) as the ciphertext.

## Defining a Decryption Round

The only difference is that a different set of tables are used (Tdn versus Ten):

```
#define ROUND_D(r, out, in0, in1, in2, in3, key) \
    out = Td0[(in0 >> 24) & 0xff] ^ \
          Td1[(in1 >> 16) & 0xff] ^ \
          Td2[(in2 >> 8) & 0xff] ^ \
          Td3[(in3 >> 0) & 0xff] ^ key \

#define LAST_D(r, out, in0, in1, in2, in3, key) \
    out = (Td4[(in0 >> 24) & 0xff] & 0xff000000) ^ \
          (Td4[(in1 >> 16) & 0xff] & 0x00ff0000) ^ \
          (Td4[(in2 >> 8) & 0xff] & 0x0000ff00) ^ \
          (Td4[(in3 >> 0) & 0xff] & 0x000000ff) ^ key \

void AES_Decrypt(const u32 rk[], u32 pt[], const u32 ct[]) {
    u32 s0, s1, s2, s3, t0, t1, t2, t3;

    s0 = ct[0] ^ rk[0];
    s1 = ct[1] ^ rk[1];
    s2 = ct[2] ^ rk[2];
    s3 = ct[3] ^ rk[3];

    ROUND_D(1, t0, s0, s3, s2, s1, rk[4]);
    ROUND_D(1, t1, s1, s0, s3, s2, rk[5]);
    ROUND_D(1, t2, s2, s1, s0, s3, rk[6]);
    ROUND_D(1, t3, s3, s2, s1, s0, rk[7]);

    ROUND_D(2, s0, t0, t3, t2, t1, rk[8]);
    ROUND_D(2, s1, t1, t0, t3, t2, rk[9]);
    ROUND_D(2, s2, t2, t1, t0, t3, rk[10]);
    ROUND_D(2, s3, t3, t2, t1, t0, rk[11]);

    ...

    ROUND_D(8, s0, t0, t3, t2, t1, rk[32]);
    ROUND_D(8, s1, t1, t0, t3, t2, rk[33]);
    ROUND_D(8, s2, t2, t1, t0, t3, rk[34]);
    ROUND_D(8, s3, t3, t2, t1, t0, rk[35]);

    ROUND_D(9, t0, s0, s3, s2, s1, rk[36]);
    ROUND_D(9, t1, s1, s0, s3, s2, rk[37]);
    ROUND_D(9, t2, s2, s1, s0, s3, rk[38]);
    ROUND_D(9, t3, s3, s2, s1, s0, rk[39]);

    LAST_D(10, pt[0], t0, t3, t2, t1, rk[40]);
    LAST_D(10, pt[1], t1, t0, t3, t2, rk[41]);
    LAST_D(10, pt[2], t2, t1, t0, t3, rk[42]);
    LAST_D(10, pt[3], t3, t2, t1, t0, rk[43]);
}
```

After the last round the state is stored in the plaintext and the decryption is complete.

## Key Expansion

Encryption relies on the two principles of *confusion* and *diffusion*. We obtain confusion by using S-boxes and diffusion by using a *different* key for each round. This expanded *key sequence* is constructed from the

original key using a special key S-box together with a bit sequence that injects a bit in a different position for each round key:

```
static const u32 rcon[10] = {
    0x01000000, 0x02000000, 0x04000000, 0x08000000,
    0x10000000, 0x20000000, 0x40000000, 0x80000000,
    0x1B000000, 0x36000000};
```

The round keys for encryption are constructed in this way:

```
void AES_KeySetupEnc(u32 rk[], const u32 Key[]) {
    int i;
    u32 temp;

    rk[0] = Key[0];
    rk[1] = Key[1];
    rk[2] = Key[2];
    rk[3] = Key[3];

    for (i=0; i<10; i++) {
        temp = rk[3];
        rk[4] = rk[0] ^ rcon[i] ^
            (Te4[(temp >> 16) & 0xff] & 0xff000000) ^
            (Te4[(temp >> 8) & 0xff] & 0x00ff0000) ^
            (Te4[(temp >> 0) & 0xff] & 0x0000ff00) ^
            (Te4[(temp >> 24) & 0xff] & 0x000000ff);
        rk[5] = rk[1] ^ rk[4];
        rk[6] = rk[2] ^ rk[5];
        rk[7] = rk[3] ^ rk[6];
        rk += 4;
    }
}
```

The round keys for decryption start with the round keys for encryption in reverse order and are then confused through the S-boxes for decryption:

```
void AES_KeySetupDec(u32 rk[], const u32 Key[]) {
    int i, j;
    u32 temp;

    AES_KeySetupEnc(rk, Key);
    /* invert the order of the round keys: */
    for (i = 0, j = 40; i < j; i += 4, j -= 4) {
        temp = rk[i]; rk[i] = rk[j]; rk[j] = temp;
        temp = rk[i+1]; rk[i+1] = rk[j+1]; rk[j+1] = temp;
        temp = rk[i+2]; rk[i+2] = rk[j+2]; rk[j+2] = temp;
        temp = rk[i+3]; rk[i+3] = rk[j+3]; rk[j+3] = temp;
    }

#define INVERT(key) \
    key = Td0[Te4[(key >> 24) & 0xff] ^ \
    Td1[Te4[(key >> 16) & 0xff] ^ \
    Td2[Te4[(key >> 8) & 0xff] ^ \
    Td3[Te4[(key >> 0) & 0xff] ^ \

    for (i = 1; i < 10; i++) {
        rk += 4;
        INVERT(rk[0]);
        INVERT(rk[1]);
        INVERT(rk[2]);
        INVERT(rk[3]);
    }
}
```

The MI-version of the key expansion goes like this:

```
SETUP-ENCRYPT-KEYS:
    CPYBLA    RK(00), KEY(01:4);
    CPYBLA    RK(01), KEY(05:4);
    CPYBLA    RK(02), KEY(09:4);
    CPYBLA    RK(03), KEY(13:4);

    CPYBTL    B, RK(03), 8, 8; XOR    L0(1:1), TE4(B), x'01';
    CPYBTL    B, RK(03), 16, 8; CPYBLA L0(2:1), TE4(B);
    CPYBTL    B, RK(03), 24, 8; CPYBLA L0(3:1), TE4(B);
    CPYBTL    B, RK(03), 0, 8; CPYBLA L0(4:1), TE4(B);
```

```

XOR      RK(04), L0      , RK(00);
XOR      RK(05), RK(04), RK(01);
XOR      RK(06), RK(05), RK(02);
XOR      RK(07), RK(06), RK(03);
...
CPYBTL   B, RK(39), 8, 8; XOR      L0(1:1), TE4(B), x'36';
CPYBTL   B, RK(39), 16, 8; CPYBLA  L0(2:1), TE4(B);
CPYBTL   B, RK(39), 24, 8; CPYBLA  L0(3:1), TE4(B);
CPYBTL   B, RK(39), 0, 8; CPYBLA  L0(4:1), TE4(B);

XOR      RK(40), L0      , RK(36);
XOR      RK(41), RK(40), RK(37);
XOR      RK(42), RK(41), RK(38);
XOR      RK(43), RK(42), RK(39);
...
INVERSION-FOR-DECRYPT:
EXCHBY   SCHEDULE(001:16), SCHEDULE(161:16);
EXCHBY   SCHEDULE(017:16), SCHEDULE(145:16);
EXCHBY   SCHEDULE(033:16), SCHEDULE(129:16);
EXCHBY   SCHEDULE(049:16), SCHEDULE(113:16);
EXCHBY   SCHEDULE(065:16), SCHEDULE(097:16);

CPYNV     RN, 4; /* SKIP FIRST (AND LAST) ROUND-KEY */
INVERT-MIX-COLS:
CPYBLA    R, RK(RN);
CPYBTL    B, R, 0,8; CPYBLA CB, TE4(B); CPYBLA RK(RN), TD0(B);
CPYBTL    B, R, 8,8; CPYBLA CB, TE4(B); XOR(S) RK(RN), TD1(B);
CPYBTL    B, R,16,8; CPYBLA CB, TE4(B); XOR(S) RK(RN), TD2(B);
CPYBTL    B, R,24,8; CPYBLA CB, TE4(B); XOR(S) RK(RN), TD3(B);
ADDN(S)   RN, 1;
CMPNV(B)  RN, 40/LO(INVERT-MIX-COLS), NLO(DONE);

```

## The Encryption/Decryption Process

There are four steps in the complete process:

- Set up the encryption key schedule with a given 16-byte key
- Encrypt a sequence of blocks of 16 bytes with the key schedule
- Set up the decryption key schedule with the same 16-byte key
- Decrypt a sequence of blocks of 16 bytes with the key schedule

We define four separate functions corresponding to each step. The **MIAES** program performs these functions as selected by the **OPERATION** field of the CONTROL parameter:

```

DCL SPCPTR .PARM1 PARM;
DCL DD     CONTROL CHAR(256) BAS(.PARM1);
DCL DD     OPERATION CHAR(8) DEF(CONTROL) POS( 1);
              /* EK      ENCRYPTION KEY SCHEDULE */
              /* E        ENCRYPT */
              /* DK      DECRYPTION KEY SCHEDULE */
              /* D        DECRYPT */
DCL DD     KEY CHAR(16) DEF(CONTROL) POS( 9);
DCL DD     SCHEDULE CHAR(176) DEF(CONTROL) POS(25);
DCL DD     RK(0:43) CHAR(4) DEF(SCHEDULE) POS(1); /* private */

```

The following two parameters are the plaintext block and the ciphertext block:

```

DCL SPCPTR .PARM2 PARM;
DCL DD     PT CHAR(16) BAS(.PARM2);

DCL SPCPTR .PARM3 PARM;
DCL DD     CT CHAR(16) BAS(.PARM3);

DCL OL PARM ( .PARM1, .PARM2, .PARM3) PARM EXT MIN(3);

```

## Operating Modes

AES is a block cipher and can be operated in two basic modes (and in very many variations on these):

- Electronic Codebook Mode (ECB)
- Cipher Block Chaining Mode (CBC)

## Electronic Codebook Mode

Electronic codebook mode is the simplest way to operate a block cipher like AES: A block of plaintext encrypts to a block of ciphertext. Since the same block of plaintext always encrypts to the same block of ciphertext it is possible (in theory at least) to create a *code book* of plaintexts and corresponding ciphertexts. With a 128-bit key, the code book will have  $2^{128}$  entries - much too large to pre-compute and store (and this is just for one key!), but the ‘codebook’ concept is still used.

A useful property of the ECB mode is that each block encrypts independently and does not depend on its context. This is important for encrypted files that must be accessed randomly, e.g. as records in a database. If the database is encrypted in ECB mode, then any record can be added or modified independently of any other record.

The problem with ECB mode is that in most real-world situations, fragments of messages tend to repeat. Different messages may have bit-sequences in common (typically at the beginning and end of messages - “Dear Mr. XXX” and “Sincerely yours...”). The decryption of Japanese and German messages – pre Enigma – in WWII was based on this concept as military messages are almost all boilerplate based and very regular. Computer generated messages, like email, have highly regular structures. Reports may have long strings of zeroes or spaces. Here is an example of a fragment of a dump of a program:

DISPLAY/ALTER/DUMP												04/12/01 19:41:00		PAGE	1														
MI PROGRAM												SUBTYPE: 01				NAME: AESCALL		ADDRESS: 1C23CCF17 000000											
SEGMENT HEADER (YYSGHDR)												TYPE 0001				SIZE 0030		NEWFLAGS 00				FLAGS 90		DOMAIN 0001		OBJECT 1C23CCF17 000000		SPACE 12CCEB45F8 001000	
EPA HEADER (YYPEAHDR)												ATT1 80				JOPT		00		TYPE 02		STYP 01							
												NAME AESCALL						SPATT 80		SPIN 00									
												UPSZ 00001000				PBAU		3F10		DVER 3600		TIME 04/12/01 19:38:16							
												SPSG 05186FC9F9 000000				ACGV		0000000000 000000		CTSG 01AA965C51 000000		OSG 1C23CCF17 001000							
												RPS2 1000				RCG2		0000		ASP 01000000		PERF 04/12/01 19:38:31							
												JPSG 0000000000 000000				CBSG		0000000000 000000		JID 00000000000000000000									
												OWAU FF1C				IPL NUM		000000031		ALIS 0000000000 000000		GRAU 0000							
												GRP 00000000000000				MAXS		0004		INFO 000000000000000000		ATT2 E0							
												COLB 01				USCNT		00000000		LEVIV 00000000 000000		USDV 0000							
												DJID 00				DENP		00000000 000000		POSG 0000000000 000000		DIRP 0000000000 000000							

Encrypted in ECB mode it looks like this

[illegible]

The structure of the plaintext “shines” through. Especially the sequence `\Ôd. ``ô> .jÜÑd!` that very likely corresponds to a string of spaces. Even the structure itself may give the attacker significant information (he might guess that it is a program dump).

A more serious problem with ECB is that the attacker could modify the encrypted message without knowing the key (or even the algorithm). Based on any structure that may be evident, the attacker could exchange 16-byte blocks and thus alter the message in a way that may be hard to detect as the blocks would decrypt to reasonable plaintext. The solution to this problem is a technique called *block chaining*.

## Cipher Block Chaining Mode

In cipher block chaining mode, the plaintext block is XORed with the previous ciphertext block (the *chaining* value) before it is encrypted. The very first plaintext block is XORed with an *initial value vector*. The initial value should be different for each message (could be a timestamp or a message identifier or some such) but does not need to be kept secret. This lack of secrecy is usually a surprise to most people; but imagine a message consisting of 100 blocks. Block number 100 is XORed with the ciphertext of block number 99; block number 99 is XORed with the ciphertext of block number 98, and so on. Thus, since the ciphertext is known, all the chaining values for blocks 100, 99, 98, etc back to and including block number 2 are known by the attacker. so not much is gained by keeping the first chaining value secret.

Decryption is completed by XORing the decrypted plaintext with the initialization vector the first time, then with the previous ciphertext block for all succeeding blocks until the end of the message.

## **The File Encryption Program, *MIFAES***

The following program (we have omitted the declarations of I/O, see Chapter 12) encrypts/decrypts a file. I use a record length of 144 bytes (a multiple of 16 bytes) for simplicity

```
DCL DD INBUF CHAR(144) BAS(.IFCB-INBUF);
DCL DD PADDED-INBUF CHAR(144);
DCL DD I BIN(4);

DCL DD OUTBUF CHAR(144) BAS(.OFCB-OUTBUF);
DCL DD PADDED-OUTBUF CHAR(144);

DCL DD MACHINE-DATA CHAR(8);
DCL DD CHAIN CHAR(16);
DCL DD SALT CHAR(16);
DCL DD SALT-FILE CHAR(10) DEF(SALT) POS(1);
DCL DD SALT-TIME CHAR(12) DEF(SALT) POS(5);

/*****/

ENTRY * (PARMS) EXT;

OPEN-INPUT-FILE:
...
OPEN-OUTPUT-FILE:
...
SET-KEY:
    CPYBLA      KEY, X'34F2A96B3C110E01770BEC45DC4828CD';

    CMPBLA(B)   PARM-GENERAL, "E"/EQ(ENCRYPT-THE-FILE);
    CMPBLA(B)   PARM-GENERAL, "D"/EQ(DECRYPT-THE-FILE);

...
ENCRYPT-THE-FILE:
    MATMDATA    MACHINE-DATA, X'0000'; /* get timestamp for initial vector */
    CPYBLA      SALT-FILE, IFCB-MEMBER;
    CVTHC       SALT-TIME, MACHINE-DATA(1:6);
    CPYBLA      CHAIN, SALT;

    CPYBLAP     OPERATION, "EK", " "; /* SETUP KEY SCHEDULE */
    CALLX       .MIAES, MIAES, *;
    CPYBLAP     OPERATION, "E ", " "; /* ENCRYPT */

GET-PLAIN-RECORD:
    CALLX       .SEPT(GET-ENTRY), GET-OPERATION, *;
    CPYBLAP     PADDED-INBUF, INBUF, " ";
    CPYNV       I, 1;
ENCRYPT-BLOCK:
    CPYBLA      PT, PADDED-INBUF(I:16);
    XOR(S)      PT, CHAIN;
    CALLX       .MIAES, MIAES, *;
    CPYBLA      OUTBUF(I:16), CT;
    CPYBLA      CHAIN, CT;
    ADDN(S)     I, 16;
    CMPNV(B)    I, 145/LO(ENCRYPT-BLOCK);

    CALLX       .SEPT(PUT-ENTRY), PUT-OPERATION, *;
    B           GET-PLAIN-RECORD;

DECRYPT-THE-FILE:
    CPYBLA      CHAIN, SALT;

    CPYBLAP     OPERATION, "DK", " "; /* SETUP KEY SCHEDULE */
    CALLX       .MIAES, MIAES, *;
    CPYBLAP     OPERATION, "D ", " "; /* DECRYPT */

GET-CIPHER-RECORD:
    CALLX       .SEPT(GET-ENTRY), GET-OPERATION, *;
    CPYNV       I, 1;
DECRYPT-BLOCK:
    CPYBLA      CT, INBUF(I:16);
    CALLX       .MIAES, MIAES, *;
    XOR(S)      PT, CHAIN;
    CPYBLA      PADDED-OUTBUF(I:16), PT;
    CPYBLA      CHAIN, CT;
```

```

ADDN(S)      I, 16;
CMPNV(B)     I, 145/LO(DECRYPT-BLOCK);

CPYBLA      OUTBUF, PADDED-OUTBUF;
CALLX      .SEPT(PUT-ENTRY), PUT-OPERATION, *;
B          GET-CIPHER-RECORD;

```

## Encryption/Decryption Speed

We tested the program on a large spoolfile, which we first copied to a database file:

```

====> CPYSPLF QPCSMPT TEXTFILE
====> CALL MIFAES PARM(TEXTFILE LSVALGAARD ENCRFILE LSVALGAARD ENCRYPT)

```

Where TEXTFILE and ENCRFILE were defined with a record length of 144 bytes and no format check. For a file with 3,678,208 bytes (25,225 records) the speed was (on a 150) as follows:

I/O	20 seconds
Call of empty MIFAES	12 seconds
Encrypt	37 seconds (100 Kb/second)

These times are pretty evenly distributed. You can improve the performance (if needed) by modifying **MIFAES** to deal with blocks longer than 16 bytes, or by using the C-version (which is up to three times faster), but it is usually not worth the effort. Decryption proceeds at the same speed as encryption.

## Compiling and Binding the C-Programs

Creating an ILE C-program is a bit more complicated than creating an OPM program. Here are the two CL-programs that I used to create the C-module **AES** and its test program **AESTEST**:

```

MKAES:
PGM
CRTCMOD MODULE(AES) SRCFILE(QCSRC) OUTPUT(*PRINT) +
        OPTION(*NOSHOWINC) DBGVIEW(*ALL)
ENDPGM

MKAESTST:
PGM
CRTCMOD MODULE(AESTEST) SRCFILE(QCSRC) OUTPUT(*PRINT) +
        OPTION(*NOSHOWINC) DBGVIEW(*ALL)
CRTPGM  PGM(AESTEST) MODULE(*CURLIB/AESTEST *CURLIB/AES) +
        ACTGRP(*CALLER)
CHGPGM  PGM(AESTEST) OPTIMIZE(*FULL)
ENDPGM

```

## The AES C-Program

Below is the mainline code from the C-version of AES operated in ECB mode:

```

#define COPY(to, from, length) for (i=0; i<length; i++) to[i] = from[i]

void AES(const char *oper, const char *key, char *schedule,
        char *plain, char *cipher) {
    u32 rK[44];
    u32 pt[4], ct[4];
    u32 *cK, *cS, *cP, *cC;
    int i;

    cK = (u32*) key;
    cS = (u32*) schedule;
    cP = (u32*) plain;
    cC = (u32*) cipher;

    if (oper[0] == 'E') {
        if (oper[1] == 'K') {
            AES_KeySetupEnc(rK, cK);
            COPY(cS, rK, 44);
        }
        else {
            COPY(rK, cS, 44);
            COPY(pt, cP, 4);
            AES_Encrypt(rK, pt, ct);
            COPY(cC, ct, 4);
        }
    }
}

```

```

    }
}
else
if (oper[0] == 'D') {
    if (oper[1] == 'K') {
        AES_KeySetupDec(rK, cK);
        COPY(cS, rK, 44);
    }
    else {
        COPY(rK, cS, 44);
        COPY(ct, cC, 4);
        AES_Decrypt(rK, pt, ct);
        COPY(cP, pt, 4);
    }
}
}
}

```

### ***The AESTEST Testprogram***

```

#include <stdio.h>
#include <string.h>

void printHex(char s[]) {
    int i;
    char ch;
    for (i=0; i<16; i++) {
        ch = (s[i]) & 0xff; printf("%2.2X", ch);
    }
    printf("\n");
}

int main(void) {
    char key [16]={0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,
                  0x08,0x09,0x0a,0x0b,0x0c,0x0d,0x0e,0x0f};
    char rkeys[176];
    char cipher[16];
    char plain [16]={0x00,0x11,0x22,0x33,0x44,0x55,0x66,0x77,
                    0x88,0x99,0xaa,0xbb,0xcc,0xdd,0xee,0xff};

    AES ("EK", key, rkeys, plain, cipher);
    AES ("E ", key, rkeys, plain, cipher);
    printHex(cipher);

    AES ("DK", key, rkeys, plain, cipher);
    AES ("D ", key, rkeys, plain, cipher);
    printHex(plain);

    getchar();
    return 0;
}

```



## Chapter 31

### The Object Information Repository

#### Extended Common Object Information

Although each object carries information about its properties in the encapsulated object itself, there is general information for many objects, such as when it was created and by whom, when it was last saved, what source file (if any) was used in creating the object, and such like. This information is stored in a record in the associated space of the context (library) into which the object is inserted. This information is still considered to belong to the object and is therefore saved/restored when the object is saved/restored.

#### Work with Object Command (**WRKOBJ**)

You can see some of that information when you issue the “Work with Object” command (**WRKOBJ**):

```
====> WRKOBJ QCMDExc
```

Display Object Description - Full				Library 1 of 1
Object . . . . .	: QCMDExc	Attribute . . . . .	:	
Library . . . . .	: QSYS	Owner . . . . .	: QSYS	
Type . . . . .	: *PGM	Primary group . . . . .	: *NONE	
User-defined information:				
Attribute . . . . .	:		:	
Text . . . . .	:		:	
Creation information:				
Creation date/time . . . . .	:	12/08/98 06:23:20	:	
Created by user . . . . .	:	*IBM	:	
System created on . . . . .	:	00000000	:	
Object domain . . . . .	:	*USER	:	
Change/Usage information:				
Change date/time . . . . .	:	07/08/99 09:07:04	:	
Usage data collected . . . . .	:	YES	:	
Last used date . . . . .	:		:	
Days used count . . . . .	:	0	:	
Reset date . . . . .	:		:	
Allow change by program . . . . .	:	NO	:	
Save/Restore information:				
Save date/time . . . . .	:		:	
Restore date/time . . . . .	:		:	
Save command . . . . .	:		:	
Device type . . . . .	:		:	

Note that **\*IBM** (as we expected) is the user that created the **QCMDExc** program object. We shall accept that **\*IBM** is not a valid user profile name. Clearly there is a way of controlling this. You may wish that *your* objects carried a distinctive creator name. Similarly, you may not want to expose the system name of your development system.

If you save or restore the object, you may see something like this:

Save/Restore information:			
Save date/time . . . . .	:	04/02/01 12:48:58	:
Restore date/time . . . . .	:		:
Save command . . . . .	:	SAVOBJ	:
Device type . . . . .	:	Save file	:
Save file . . . . .	:	LSVALGAARD/S	:

The question is: how much of this is under your control? Can you change or reset this information? The present Chapter explores that subject.

## The OIR (Object Information Repository)

Our investigation starts with the library object. Here is the object header for my library LSVALGAARD (object type x'0401'):

```
Address 01AA965C51 000000
000000 00010058 00908000 01AA965C 51000000  ...i.°0..j o*é...
000010 40010000 00000000 29387B2E 9C000020  ... ..#..æ... ← Associated space
000020 80000401 D3E2E5C1 D3C7C1C1 D9C44040 0.. LSVALGAARD
000030 40404040 40404040 40404040 40404040  ....\...°....
000040 40400000 00001FE0 00000090 3F100601  a¶dAG00...«...
000050 81B684C1 C7CD8000 061A0537 8A000000
```

Here is the beginning of the primary associated space:

```
Address 29387B2E9C 000020
000020 00000010 00000003 08004040 00000000  ....
000030 00000000 00000000 01AA965C 51000400  ... ..j o*é... Li brary
000040 00008000 00000000 385444D3 32000E7F  ..0.. ..èaL... " Index
000050 00008000 00000000 1FE229A5 7D00197F  ..0.. ..S.v'... " Space
000060 00000000 00000000 00000000 00000000  ....
000070 00000000 00000000 00000000 00000000  ....
000080 00000000 00000000 00000000 00000000  ....
```

We see system pointers to a library (just LSVALGAARD again), to an index and to a space. A different view is afforded by the “Dump System Object” command (DMPSYSOBJ):

===> DUMPSYSOBJ OBJ(LSVALGAARD) CONTEXT(\*MCHCTX) TYPE(04)

```
DMPSYSOBJ PARAMETERS
OBJ- LSVALGAARD
TYPE- *ALL SUBTYPE- *ALL
OBJECT TYPE- CONTEXT
NAME- LSVALGAARD TYPE- 04 SUBTYPE- 01
LIBRARY- MACHIN E CONTEXT
CREATION- 12/14/00 18:46:07
OWNER- QSECOFR
ATTRIBUTES- 0000
ADDRESS- 01AA965C51 000000
CONTEXT-
000000 00FF0000 00007950 0401D3E2 E5C1D3C7 C1C1D9C4 40404040 40404040 40404040 * 0 ` & LSVALGAARD *
000020 40404040 40404040 80000000 00000000 00001FE0 00000000 00000000 00000000 *
000040 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *
OBJECTS-
02 01 A 01055491AB 000000
... 19 15 MICHLP1 154ADA733B 000000
PRIMARY ASSOCIATED SPACE-
000000 00000010 00000003 08004040 00000000 00000000 00000000 01AA965C 51000400 *
000020 00008000 00000000 385444D3 32000E7F 00008000 00000000 1FE229A5 7D00197F * èaL " i o*é *
000040 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 * S v' " *
LIBR ES 000060 TO 0001FF SAME AS ABOVE
```

We recognize the system pointers to the index (type/subtype x'0E90', \*QDIDX) and to the data space (type/subtype x'1952', \*OIRS):

```
. POINTERS-
000010 SYP 04 01 LSVALGAARD MACHIN E CONTEXT 0000 0000 *LIB
000020 SYP 0E 90 LSVALGAARD 7F10 0000 *QDIDX
000030 SYP 19 52 LSVALGAARD 7F10 0000 *OIRS

OBJECT TYPE- INDEX
NAME- LSVALGAARD TYPE- 0E SUBTYPE- 90
CREATION- 12/14/00 18:46:07
OWNER- QSECOFR
ATTRIBUTES- 0800
ADDRESS- 385444D332 000000
INDEX ATTRIBUTES-
000000 00FF0000 00000071 0E90D3E2 E5C1D3C7 C1C1D9C4 40404040 40404040 40404040 * 0 É °LSVALGAARD *
000020 40404040 40404040 80000000 00000000 00000000 00000000 00000000 00000000 *
000040 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *
000060 64001800 0C000015 40000013 E4000001 A7 *A U x *
```

The index entries have a length of 24 bytes (x'0018') and a key length of 12 bytes (x'000C'):

```
INDEX ENTRIES-
000001-
000000 0201C140 40404040 40404040 00000001 00000006 00000000 * (A) <= name of object *
000002-
000000 0201C1C1 C1404040 40404040 00000001 00000007 00000000 * AAA *
```

The “boxed” number is the offset to a 512-byte area within the data space where the extended object description data is stored. This area is called the *OIR* (Object Information Repository) for the object. The offset is expressed as an ordinal number where the first entry is number 1; the second entry is number 2, etc. The space starts at byte-offset 256, so to convert the entry ordinal number to a byte offset within the space object (the *relative* offset), we multiply the ordinal number by 512, then subtract 256.

Stored in the binary word just before the offset is a “record count” giving the number of 512-byte records that make up the OIR for this entry.

```

OBJECT TYPE- SPACE *OIRS
NAME- LSVALGAARD TYPE- 19 SUBTYPE- 52
CREATION- 12/14/00 18: 46: 07 SIZE- 0000036000
OWNER- QSECOFR ADDRESS- 08 SUBTYPE- 01
ATTRIBUTES- 0800
SPACE ATTRIBUTES-
000000 00FFFF00 00000074 1952D3E2 E5C1D3C7 C1C1D9C4 40404040 40404040 40404040 * 0 È eLSVALGAARD *
000020 40404040 40404040 C0000000 00000000 00035F00 00000000 00000000 00000000 * { *
000040 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 * *
000060 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 * *
SPACE-
000000 00000000 00000000 00000000 00000000 80000000 00000000 29387B2E 9C000200 * # æ *
000020 00035F00 00360000 00000000 00000000 00000000 00000000 00000000 00000000 * *
000040 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 * *
LINES 000060 TO 0000FF SAME AS ABOVE
000100 FFFFFFFF 0401D3E2 E5C1D3C7 C1C1D9C4 00000001 000BD7D9 D6C44040 40404040 * LSVALGAARD PROD *
000120 F0000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *0 *
000140 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 * *
LINES 000160 TO 0001FF SAME AS ABOVE
...

```

Here is the entry for the program object A (the relative offset is x'B00' = 6\*512 - 256):

```

000B00 FFFFFFFF 0201C140 40404040 40404040 00000001 00144040 40404040 40404040 * A *
000B20 40404040 40404040 40400032 E3C5E7E3 40404040 40404040 40404040 40404040 * TEXT *
000B40 40404040 40404040 40404040 40404040 40404040 404040C5 D5C40082 * END b *
000B60 0012823F 40392694 00000000 00000000 00001001 00006000 00000000 00000000 * b m - *
000B80 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 * *
000BA0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 * S LSVALGAARD *
000BC0 00000000 0000E240 40404040 40404040 D3E2E5C1 D3C7C1C1 D9C40000 00000000 * *
000BE0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 * *
000C00 000000A8 F9F9F9F9 E7E7F1E5 F4D9F4D4 E7D6C2D1 C3E3D3E5 D3F14040 40404040 * y9999XX1V4R4MX0BJCTLVL1 *
000C20 4040F5F7 F6F9E2E2 F1E5F4D9 F4D4F0E5 F4D9F4D4 F0F1F0F1 F0F3F2F4 F1F7F2F1 * 5769SS1V4R4MOV4R4M010103241721 *
000C40 F2F9F1F0 F1F0F1F0 F2F0F3F0 F4F0F5D8 D4C9E2D9 C3404040 40D4E8C8 C5D3D7C5 * 291010102030405QMI SRC MYHELPE *
000C60 D94040D4 E8D3C9C2 40404040 40C1D7C1 D9C3C4C6 60D3E5D3 40404040 00000000 *R MYLI B APARCDF-LVL *
000C80 00000000 D3C4C9C4 D3C4D6D7 00000000 00000000 00000000 D7E3C3D4 * LDIDL DOP PTCM *
000CA0 C9C40000 00000000 00000000 001AD3E2 E5C1D3C7 C1C1D9C4 D4E8C1E2 F4F0F040 *ID LSVALGAARDMYAS400 *
000CC0 82539917 174F0000 002E40E8 E4E2C5D9 C4C5C6C1 E3D94040 40404040 40404040 *ber | YUSERDEFATR *
000CE0 40404040 40404040 40404040 40404040 40404040 40404040 00000000 00000000 *

```

The *absolute* offset is x'C00', so here is a better look:

```

000C00 FFFFFFFF 0201C140 40404040 40404040 ..... A
000C10 00000001 00144040 40404040 40404040 .....
000C20 40404040 40404040 40400032 E3C5E7E3 .. TEXT
000C30 40404040 40404040 40404040 40404040
000C40 40404040 40404040 40404040 40404040
000C50 40404040 40404040 404040C5 D5C40082 END b
000C60 0012823F 40392694 00000000 00000000 .. b .. m ..
000C70 00001001 00006000 00000000 00000000 .....
000C80 00000000 00000000 00000000 00000000 .....
000C90 00000000 00000000 00000000 00000000 .....
000CA0 00000000 00000000 00000000 00000000 .....
000CB0 00000000 00000000 00000000 00000000 .....
000CC0 00000000 0000E240 40404040 40404040 ..... S
000CD0 D3E2E5C1 D3C7C1C1 D9C40000 00000000 LSVALGAARD ..
000CE0 00000000 00000000 00000000 00000000 .....
000CF0 00000000 00000000 00000000 00000000 .....
000D00 000000A8 F9F9F9F9 E7E7F1E5 F4D9F4D4 ... y9999XX1V4R4M
000D10 E7D6C2D1 C3E3D3E5 D3F14040 40404040 X0BJCTLVL1
000D20 4040F5F7 F6F9E2E2 F1E5F4D9 F4D4F0E5 5769SS1V4R4MOV
000D30 F4D9F4D4 F0F1F0F1 F0F3F2F4 F1F7F2F1 4R4M010103241721
000D40 F2F9F1F0 F1F0F1F0 F2F0F3F0 F4F0F5D8 291010102030405Q
000D50 D4C9E2D9 C3404040 40D4E8C8 C5D3D7C5 MI SRC MYHELPE
000D60 D94040D4 E8D3C9C2 40404040 40C1D7C1 R MYLI B APA
000D70 D9C3C4C6 60D3E5D3 40404040 00000000 RCDF-LVL ....
000D80 00000000 D3C4C9C4 D3C4D6D7 00000000 .... LDIDL DOP ...
000D90 00000000 00000000 00000000 D7E3C3D4 ..... PTCM
000DA0 C9C40000 00000000 00000000 001AD3E2 ID ..... LS
000DB0 E5C1D3C7 C1C1D9C4 D4E8C1E2 F4F0F040 VALGAARDMYAS400
000DC0 82539917 174F0000 002E40E8 E4E2C5D9 ber .. | ... YUSER
000DD0 C4C5C6C1 E3D94040 40404040 40404040 DEFATR
000DE0 40404040 40404040 40404040 40404040
000DF0 40404040 40404040 00000000 00000000 .....

```

We shall decipher the contents of the OIR in a later section of this chapter. Let's first write a program, **MI WRK OIR**, to retrieve (and optionally change) the entire OIR-record.

### Work with OIR (MI WRK OIR)

To maintain maximum flexibility, the **MI WRK OIR** program is an API. You call it with a qualified object name (10-character object name followed by 10-character library name), an object type and subtype, and a 512-character OIR-receiver area. If an optional fourth parameter, a 512-character new OIR-record, is

present, it replaces the OIR-record. It is worth noting that the OIR is an OS/400 concept, not an MI-object, hence the limitation of 10-character names.

```
DCL SPCPTR . PARM1 PARM;
DCL DD      PARM-NAME CHAR(20) BAS(. PARM1);
DCL DD      OBJ-NAME  CHAR(10) DEF(PARM-NAME) POS( 1);
DCL DD      LIB-NAME  CHAR(10) DEF(PARM-NAME) POS(11);

DCL SPCPTR . PARM2 PARM;
DCL DD      PARM-TYPE CHAR(2) BAS(. PARM2);

DCL SPCPTR . PARM3 PARM;
DCL DD      PARM-OIR CHAR(512) BAS(. PARM3);

DCL SPCPTR . PARM4 PARM;
DCL DD      PARM-OIR-NEW CHAR(512) BAS(. PARM4);

DCL OL PARAMETER-LIST(. PARM1, . PARM2, . PARM3, . PARM4) PARM EXT MIN(3);
DCL DD PARM-LENGTH BIN(2);
```

System pointers to the object index and the OIR data space resides in the associated space for the library:

```
DCL SYSPTR . LIBSYS;
DCL SPCPTR . LIBSPC;
DCL DD      LIBSPC CHAR(512) BAS(. LIBSPC);
DCL SYSPTR . SYS-IDX POS(33) DEF(LIBSPC);
DCL SYSPTR . SYS-OIR POS(49) DEF(LIBSPC);

DCL SPCPTR . SPC-OIR;
DCL DD      SPC-OIR CHAR(16000000) BAS(. SPC-OIR);

DCL SPCPTR . OIR-DATA;
DCL DD      OIR-DATA CHAR(512) BAS(. OIR-DATA);
```

We remember from Chapter 25 how to retrieve index entries:

```
DCL SPCPTR . INDEX-OPT INIT(INDEX-OPT);
DCL DD INDEX-OPT CHAR(18) BDRY(16);
DCL DD INDEX-RULE-OPTION BIN(2) DEF(INDEX-OPT) POS( 1) INIT(1);
DCL DD INDEX-ARG-LENGTH BIN(2) DEF(INDEX-OPT) POS( 3) INIT(12);
DCL DD INDEX-ARG-OFFSET  BIN(2) DEF(INDEX-OPT) POS( 5) INIT(0);
DCL DD INDEX-OCR-COUNT   BIN(2) DEF(INDEX-OPT) POS( 7) INIT(1);
DCL DD INDEX-RTN-COUNT   BIN(2) DEF(INDEX-OPT) POS( 9) INIT(0);
DCL DD INDEX-ENTRY(2)    CHAR(4) DEF(INDEX-OPT) POS(11);

DCL SPCPTR . INDEX-SEARCH-ARG INIT(INDEX-SEARCH-ARG);
DCL DD      INDEX-SEARCH-ARG CHAR(12) BDRY(16);

DCL SPCPTR . INDEX-DATA INIT(INDEX-DATA);
DCL DD      INDEX-DATA CHAR(24) BDRY(16);
DCL DD      INDEX-KEY   CHAR(12) DEF(INDEX-DATA) POS( 1);
DCL DD      INDEX-COUNT BIN( 4) DEF(INDEX-DATA) POS(13);
DCL DD      INDEX-OFFSET BIN( 4) DEF(INDEX-DATA) POS(17);
DCL DD      *           CHAR( 4) DEF(INDEX-DATA) POS(21);

DCL DD OFFSET BIN(4);
```

We need the Resolve System Pointer template to get a pointer to the library (type x'0401') specified:

```
DCL DD RESOLVE CHAR(34) BDRY(16);
DCL DD RESOLVE-TYPE CHAR( 2) POS( 1) DEF(RESOLVE) INIT(X' 0401' );
DCL DD RESOLVE-NAME CHAR(30) POS( 3) DEF(RESOLVE);
DCL DD *           CHAR( 2) POS(33) DEF(RESOLVE) INIT(X' 0000' );

ENTRY *(PARAMETER-LIST) EXT;
CPYBLAP RESOLVE-NAME, LIB-NAME, " ";
RSLVSP . LIBSYS, RESOLVE, *, *;
```

Obtain a space pointer to the OIR space:

```
SETSPFP . LIBSPC, . LIBSYS;
SETSPFP . SPC-OIR, . SYS-OIR;
```

The key for the index is the object type/subtype followed by the object name. We set the key in the search argument variable, the issue a “Find Independent Index entry” instruction to find 1 entry with that key (the rule being 1 = “equal”):

```

RETRIEVE-OIR-DATA:
  CPYBLA      INDEX-SEARCH-ARG, PARM-TYPE;
  CPYBLA      INDEX-SEARCH-ARG(3:10), OBJ-NAME;
  FNDINDEX    INDEX-DATA, .SYS-IDX, .INDEX-OPT, .INDEX-SEARCH-ARG;
  CMPNV(B)    INDEX-RTN-COUNT, O/HI (FOUND-OIR);

```

If the return count is zero, the object was not found and we return an OIR-record that is all blanks:

```

  CPYBREP      PARM-OIR, " ";
  B           DONE;

```

If the object was found, we compute the relative offset to the OIR-record, add the offset to the OIR space pointer and can then return the OIR data:

```

FOUND-OIR:
  MULT      OFFSET, INDEX-OFFSET, 512;
  SUBN(S)   OFFSET, 256; /* make it a relative offset */
  ADDSPP    OIR-DATA, .SPC-OIR, OFFSET;
  CPYBLA    PARM-OIR, OIR-DATA;

```

If a fourth parameter is present, its contents replace the OIR-record:

```

  STPLEN      PARM-LENGTH;
  CMPNV(B)    PARM-LENGTH, 4/LO(DONE);
  CPYBLA      OIR-DATA, PARM-OIR-NEW;
DONE:
  RTX        *;

```

It is straightforward to extend the API to also retrieve further OIR records should the INDEX-COUNT be greater than 1.

## Must be System State

Since **MI WRKOIR** uses the **SETSPFPF** instruction and since the OIR is in the system domain, **MI WRKOIR**, must have the system state attribute.

## The Change Object Description API (QLI COBJD)

Some of the fields in the OIR can be changed by the “Change Object Description” (**QLI COBJD**) API. Before the object can be changed with the API, the “allow change by program field” within the OIR is checked. If this field is set to “N”, the API refuses to change anything. When an object has been updated successfully by the QLI COBJD API, two fields within the OIR are updated: “the date and time the object was changed by the API” and “changed by program”.

Needless to say, the system cannot prevent you from changing the OIR using the method described in this chapter.

## Fields in the OIR

The following data declaration identifies the fields as far as I have been able to determine:

```

DCL SPCPTR .OIR INIT(OIR);
DCL DD .OIR CHAR(512);
DCL DD OIR-RECORD-ID          BIN(4) DEF(OIR) POS( 1);
DCL DD OIR-OBJ-TYPE           CHAR(2) DEF(OIR) POS( 5);
DCL DD OIR-OBJ-NAME           CHAR(10) DEF(OIR) POS( 7);
DCL DD OIR-NBR-OF-RECS        BIN(4) DEF(OIR) POS(17);
DCL DD *                      BIN(2) DEF(OIR) POS(21);
DCL DD OIR-ATTRIBUTE          CHAR(10) DEF(OIR) POS(23);
DCL DD *                      CHAR(4) DEF(OIR) POS(33);
DCL DD *                      BIN(4) DEF(OIR) POS(37);
DCL DD OIR-TEXT-SIZE          BIN(2) DEF(OIR) POS(43);
DCL DD OIR-TEXT               CHAR(50) DEF(OIR) POS(45);
DCL DD *                      BIN(2) DEF(OIR) POS(95);
DCL DD *                      BIN(2) DEF(OIR) POS(97);
DCL DD OIR-SAVE-TIME           CHAR(8) DEF(OIR) POS(99);
DCL DD OIR-RESTORE-TIME        CHAR(8) DEF(OIR) POS(107);
DCL DD *                      CHAR(4) DEF(OIR) POS(115);
DCL DD *                      BIN(2) DEF(OIR) POS(119);
DCL DD *                      CHAR(78) DEF(OIR) POS(121);
DCL DD OIR-SAVE-FILE           CHAR(10) DEF(OIR) POS(199);
DCL DD OIR-SAVE-LIBRARY        CHAR(10) DEF(OIR) POS(209);

```

```

DCL DD *                               CHAR(40) DEF(OIR) POS(219);
DCL DD *                               BIN(2) DEF(OIR) POS(259);
DCL DD OIR-LICENSED-PROGRAM             CHAR(13) DEF(OIR) POS(261);
DCL DD OIR-OBJ-CONTROL-LEVEL            CHAR(8) DEF(OIR) POS(274);
DCL DD *                               CHAR(1) DEF(OIR) POS(282);
DCL DD *                               CHAR(8) DEF(OIR) POS(283);
DCL DD OIR-REFERENCE-COMPILER           CHAR(13) DEF(OIR) POS(291);
DCL DD OIR-VERSION-RELEASE              CHAR(6) DEF(OIR) POS(304);
DCL DD OIR-CREATION-DATE                 CHAR(13) DEF(OIR) POS(310);
DCL DD OIR-REF-SOURCE-CHANGED           CHAR(13) DEF(OIR) POS(323);
DCL DD OIR-REF-SOURCE-FILE              CHAR(10) DEF(OIR) POS(336);
DCL DD OIR-REF-SOURCE-MEMBER            CHAR(10) DEF(OIR) POS(346);
DCL DD OIR-REF-SOURCE-LIBRARY           CHAR(10) DEF(OIR) POS(356);
DCL DD OIR-APAR-CODE                    CHAR(6) DEF(OIR) POS(366);
DCL DD OIR-PTF-LEVEL-2                  CHAR(5) DEF(OIR) POS(372);
DCL DD *                               CHAR(4) DEF(OIR) POS(377);
DCL DD *                               CHAR(8) DEF(OIR) POS(381);
DCL DD OIR-PROD-LANGUAGE-ID             CHAR(4) DEF(OIR) POS(389);
DCL DD OIR-PROD-OPTION-ID               CHAR(4) DEF(OIR) POS(393);
DCL DD *                               CHAR(16) DEF(OIR) POS(397);
DCL DD OIR-PTF-LEVEL-1                  CHAR(2) DEF(OIR) POS(413);
DCL DD OIR-COMPONENT-ID                 CHAR(4) DEF(OIR) POS(415);
DCL DD *                               CHAR(12) DEF(OIR) POS(419);
DCL DD OIR-CREATING-USER                 CHAR(10) DEF(OIR) POS(431);
DCL DD OIR-CREATING-SYSTEM              CHAR(8) DEF(OIR) POS(441);
DCL DD OIR-WHEN-CHANGED-BY-API           CHAR(8) DEF(OIR) POS(449);
DCL DD *                               BIN(2) DEF(OIR) POS(457);
DCL DD OIR-ALLOW-USER-CHANGE            CHAR(1) DEF(OIR) POS(459);
DCL DD OIR-CHANGED-BY-PGM                CHAR(1) DEF(OIR) POS(460);
DCL DD OIR-USER-DEFINED-ATTR             CHAR(10) DEF(OIR) POS(461);
DCL DD OIR-SORT-SEQUENCE                 CHAR(10) DEF(OIR) POS(471);
DCL DD OIR-SORT-LIBRARY                  CHAR(10) DEF(OIR) POS(481);
DCL DD OIR-LANGUAGE-ID                  CHAR(10) DEF(OIR) POS(491);
DCL DD *                               CHAR(12) DEF(OIR) POS(501);

```

**Record ID:** Is a zero for an unused entry. The first record for an entry with data has a record ID of -1, the second a record ID of -2, etc. Note that these numbers are negative.

**Object type and name:** The type and subtype in their hexadecimal representation (e.g. x'0201' for a program). The name is padded with blanks to 10 characters. Note that this restricts the object to be an OS/400 object.

**Number of records:** The number of 512-byte records for this object.

**Attribute:** The extended attribute for the object (e.g. CLP for CL-programs, SAVF for save files).

**Text and text size:** The text field always contains 50 characters. The size field may specify a smaller number (even zero), but the text is always padded with blanks to 50 characters.

**Save and Restore times:** Standard 8-byte machine clock timestamps of when the object was last saved and restored. If the object has not yet saved or restored the timestamps are binary zeroes.

**Save file and library:** The name of the save file and the library where it is stored.

**Licensed program:** The name, version level, release level, and modification level of the licensed program. Objects that are a part of an IBM licensed program have a valid licensed program name (7 characters containing 0-9 and uppercase A-Z). The version field is in the VxRxMy format.

**Object control level:** The object control level for the object. IBM programs will have an 8-character decimal value (if any).

**Reference compiler:** The name, version level, release level, and modification level of the compiler. Objects created with IBM products will have the licensed program name of the compiler in the compiler name field and a version field in the VxRxMy format.

**Version/Release:** Version/Release/Modification level in the VxRxMy format where x must be 0-9 and y must be in 0-9 or in A-Z. If you want to conform to the system, this format should be followed.

**Creation date:** The date and time when the object was created in the silly IBM format CYYMMDDHHMMSS, where C=0 for the 20<sup>th</sup> century and C=1 for the 21<sup>st</sup> century.

**Source changed:** The date and timestamp of the source (from which the object was created) in the 13-character date format CYYMMDDHHMMSS.

**Source file, member, and library:** Identification of the source from which the object was created.

**APAR code:** The Authorized Program Analysis Report identification that caused this object to be patched. IBM APARs have an uppercase alphabetic character followed by 5 decimal numbers.

**PTF level:** The program temporary fix (PTF) that resulted in the creation of the object. For IBM objects the first 2 characters are a prefix ID and the remaining 5 characters are the program change ID (decimal). The field is blank if the object was not changed because of a PTF. Note that this is a *split* field.

**Product Language ID:** The language identifier associated with the object. Objects that are a part of an IBM licensed program must have one of the allowed languages in the 29xx format.

**Product option ID:** Identifies part of a licensed program (product). Products can have multiple options. Objects that are a part of an IBM licensed program must be 0000 (\*BASE) through 0099.

**Component ID:** The product administrator owns this field. It can be used to track information about objects at a lower level than the product option ID.

**Creating user:** The user profile of the user who created the object. IBM product use \*IBM.

**Creating system:** The name of the system where the object was first created. IBM products use 00000000.

**When changed by API:** The timestamp when the OIR was last changed by the QLI COBJD API.

**Allow user change:** If this field contains an “N”, the QLI COBJD API cannot be used to change the OIR. To allow changes the field should be set to a blank (not a “Y”).

**Changed by program:** If the QLI COBJD API was ever used to change the OIR this field contains a “Y”, otherwise it is blank.

**User-defined attribute:** An attribute you define. This should not be confused with the extended attribute that is set by the system when the object is created.

**Sort sequence and library:** Either \*HEX or a sort table and the library where to find it.

**Language ID:** Usually \*JOB RUN. Can be a three-letter language abbreviation (*e.g.* ENU for US English).

## The **MI SHWOI R** Test Program

To show working with the OIR in action, we write a simple interactive program (**MI SHWOI R**) to show and change selected data items from the OIR of a given object. Here is the screen display:

Work with OIR		
Object name . . . . .	<u>A</u>	Name
Library . . . . .	<u>LSVALGAARD</u>	Name
Object type/subtype . . . . .	<u>Q201</u>	Hexadecimal values
Creating user . . . . .	<u>LSVALGAARD</u>	User profile
Creating system . . . . .	<u>MYAS400</u>	System name
Allow changes . . . . .	—	N=No, blank=Yes
Changed by program . . . . .	—	blank=No, Y=Yes
User defined attribute . . . . .	_____	Characters
PTF level . . . . .	_____	PP12345
Version/Release . . . . .	<u>V4R4M0</u>	VxRxMy
Product language ID . . . . .	_____	Language ID
Product option ID . . . . .	_____	Option ID
Component ID . . . . .	_____	Component ID
Attribute . . . . .	<u>MI</u>	Characters
Last save date/time . . . . .	<u>0000000000000000</u>	Hex timestamp
Last restore date/time . . . . .	<u>0000000000000000</u>	Hex timestamp
Creation date/time . . . . .	<u>20010422082552</u>	Date/time
Text description . . . . .		
F3=Exit F11=Change F12=Cancel		

The arguments for **MI WRKOI R** are:

```
DCL SPCPTR .OBJECT INIT(OBJECT);
DCL DD      OBJECT CHAR(20);
      DCL DD OBJ-NAME  CHAR(10) POS( 1) DEF(OBJECT); /* Name */
      DCL DD LIB-NAME  CHAR(10) POS(11) DEF(OBJECT); /* Lib*/

DCL SPCPTR .TYPE INIT(TYPE);
DCL DD      TYPE CHAR( 2);

DCL SPCPTR .OIR INIT(OIR);
DCL DD      OIR CHAR(512);
      DCL DD OIR-RECORD-ID          BIN(4) DEF(OIR) POS( 1);
      DCL DD OIR-OBJ-TYPE          CHAR(2) DEF(OIR) POS( 5);
      DCL DD OIR-OBJ-NAME          CHAR(10) DEF(OIR) POS( 7);
... /* as above */

DCL OL MIWRKoir(.OBJECT, .TYPE, .OIR) ARG;

DCL SPCPTR .NEW INIT(NEW);
DCL DD      NEW CHAR(512);
DCL OL MINewOIR(.OBJECT, .TYPE, .OIR, .NEW) ARG;
```

We'll use our standard screen handler from Chapter 18:

```
ENTRY * EXT;
  CPYBLA      RESOLVE-TYPE, X' 0201' ;
  CPYBLAP     RESOLVE-NAME, "MIWRKoir", " ";
  RSLVSP      .MIWRKoir, RESOLVE, *, *;
  CPYBLAP     RESOLVE-NAME, "MI SCRNI O", " ";
  RSLVSP      .MI SCRNI O, RESOLVE, *, *;

  CPYBLA      CTRL-OPCODE, "OPEN";
  CALLX       .MI SCRNI O, MI SCRNI O, *;

SHOW-THE-SCREEN:
  CPYBLA      A-F11, "L": /* "gray out" function key */
  CPYBREP     CTRL-CURSOR-POSITION, "0";
  CPYBLA      CTRL-OPCODE, "WRITE";
  CALLX       .MI SCRNI O, MI SCRNI O, *;

  CPYBREP     S-MESSAGE, " ";
  CPYBLA      A-MESSAGE, "B";

GET-THE-SCREEN:
  CPYBLA      CTRL-OPCODE, "READ";
  CALLX       .MI SCRNI O, MI SCRNI O, *;

  CMPNV(B)    CTRL-CMD-KEY, 0/EQ(WORK-WITH-DATA);
  CMPNV(B)    CTRL-CMD-KEY, 3/EQ(DONE);
  CMPNV(B)    CTRL-CMD-KEY, 12/EQ(DONE);

FUNCTION-KEY-NOT-USED:
  CPYBLAP     S-MESSAGE, "Function key not used", " ";
SHOW-ERROR-MESSAGE:
  CPYBLA      A-MESSAGE, "T";
  B           SHOW-THE-SCREEN;
```

We now have a screen from the user. Check that the object is specified:

```
WORK-WITH-DATA:
  CMPBLAP(B)  S-OBJECT, " ", " " /EQ(SUPPLY-OBJECT);
  CMPBLAP(B)  S-LIBRARY, " ", " " /EQ(SUPPLY-LIBRARY);
  CMPBLAP(B)  S-TYPE, " ", " " /EQ(SUPPLY-TYPE);

  CPYBLA      RESOLVE-TYPE, X' 0401' ;
  CPYBLAP     RESOLVE-NAME, S-LIBRARY, " ";
  RSLVSP      .LIBRARY, RESOLVE, *, *;

  CPYBLA      OBJ-NAME, S-OBJECT;
  CPYBLA      LIB-NAME, S-LIBRARY;
  CVTCH       TYPE, S-TYPE;
  CALLX       .MIWRKoir, MIWRKoir, *;

  CMPBLAP(B)  OIR, " ", " " /EQ(OBJECT-NOT-FOUND);
```

If the OIR returned by MI WRKOI R was all spaces, the object is not found. Otherwise we have valid data and we put some of it on screen (as Jean-Luc Picard would say):



SHOW-THE-OIR:

CPYBLA	S-CREATING-USER,	OIR-CREATING-USER;
CPYBLA	S-CREATING-SYSTEM,	OIR-CREATING-SYSTEM;
CPYBLA	S-ALLOW-USER-CHANGE,	OIR-ALLOW-USER-CHANGE;
CPYBLA	S-CHANGED-BY-PGM,	OIR-CHANGED-BY-PGM;
CPYBLA	S-USER-DEFINED-ATTR,	OIR-USER-DEFINED-ATTR;
CPYBLA	S-PTF-LEVEL(1: 2),	OIR-PTF-LEVEL-1;
CPYBLA	S-PTF-LEVEL(3: 5),	OIR-PTF-LEVEL-2;
CPYBLA	S-VERSION-RELEASE,	OIR-VERSION-RELEASE;
CPYBLA	S-TEXT,	OIR-TEXT;
CPYBLA	S-PROD-LANGUAGE-ID,	OIR-PROD-LANGUAGE-ID;
CPYBLA	S-PROD-OPTION-ID,	OIR-PROD-OPTION-ID;
CPYBLA	S-COMPONENT-ID,	OIR-COMPONENT-ID;
CPYBLA	S-ATTRIBUTE,	OIR-ATTRIBUTE;

Dealing with the various timestamps and dates is slightly more complex. We could have used the conversion code from Chapter 4, but as our goal here is to investigate the OIR rather than to show date conversions, we simply convert to hex instead:

CMPBLAP(B)	OIR-SAVE-TIME, " ", " "/NEQ(=+2);
CPYBREP	S-SAVE-TIME, " ";
CMPBLAP(B)	OIR-SAVE-TIME, " ", " "/EQ(=+2);
CVTHC	S-SAVE-TIME, OIR-SAVE-TIME;
CMPBLAP(B)	OIR-RESTORE-TIME, " ", " "/NEQ(=+2);
CPYBREP	S-RESTORE-TIME, " ";
CMPBLAP(B)	OIR-RESTORE-TIME, " ", " "/EQ(=+2);
CVTHC	S-RESTORE-TIME, OIR-RESTORE-TIME;

Here we deal with the “century” problem. I hate to think of the hundred of thousands of lines of code that could have been saved if IBM had simply stored the century as is, rather than trying to save a byte:

CPYBLA	S-CREATON-DATE(1: 2), "20";
CMPBLA(B)	OIR-CREATON-DATE(1: 1), "0"/NEQ(=+2);
CPYBLA	S-CREATON-DATE(1: 2), "19";
CPYBLA	S-CREATON-DATE(3: 12), OIR-CREATON-DATE(2: 12);
CPYBLA	A-F11, "U": /* Enable function key */
CPYBLA	CTRL-CURSOR-POSITION, "00000";
CPYBLA	CTRL-OPCODE, "WRITE";
CALLX	.MI SCRNI O, MI SCRNI O, *;
CPYBLA	CTRL-OPCODE, "READ";
CALLX	.MI SCRNI O, MI SCRNI O, *;
CMPNV(B)	CTRL-CMD-KEY, 0/EQ(WORK-WITH-DATA);
CMPNV(B)	CTRL-CMD-KEY, 3/EQ(DONE);
CMPNV(B)	CTRL-CMD-KEY, 12/EQ(DONE);
CMPNV(B)	CTRL-CMD-KEY, 11/NEQ(FUNCTION-KEY-NOT-USED);

If function-key 11 was pressed, change the OIR to the values given. No detailed checking of the input:

CHANGE-THE-OIR:

CPYBLA	OIR-CREATING-USER,	S-CREATING-USER;
CPYBLA	OIR-CREATING-SYSTEM,	S-CREATING-SYSTEM;
CPYBLA	OIR-ALLOW-USER-CHANGE,	S-ALLOW-USER-CHANGE;
CPYBLA	OIR-CHANGED-BY-PGM,	S-CHANGED-BY-PGM;
CPYBLA	OIR-USER-DEFINED-ATTR,	S-USER-DEFINED-ATTR;
CPYBLA	OIR-PTF-LEVEL-1,	S-PTF-LEVEL(1: 2);
CPYBLA	OIR-PTF-LEVEL-2,	S-PTF-LEVEL(3: 5);
CPYBLA	OIR-VERSION-RELEASE,	S-VERSION-RELEASE;
CPYBLA	OIR-TEXT,	S-TEXT;
CPYBLA	OIR-PROD-LANGUAGE-ID,	S-PROD-LANGUAGE-ID;
CPYBLA	OIR-PROD-OPTION-ID,	S-PROD-OPTION-ID;
CPYBLA	OIR-COMPONENT-ID,	S-COMPONENT-ID;
CPYBLA	OIR-ATTRIBUTE,	S-ATTRIBUTE;
CVTCH	OIR-SAVE-TIME,	S-SAVE-TIME;
CVTCH	OIR-RESTORE-TIME,	S-RESTORE-TIME;
CPYBLA	OIR-CREATON-DATE(1: 1), "1";	
CMPBLA(B)	S-CREATON-DATE(1: 2), "20"/EQ(=+2);	
CPYBLA	OIR-CREATON-DATE(1: 1), "0";	
CPYBLA	OIR-CREATON-DATE(2: 12), S-CREATON-DATE(3: 12);	
CPYBLA	NEW, OIR;	
CALLX	.MI WRKOIR, MI NEWOIR, *;	
B	WORK-WITH-DATA;	

Here we deal with various error conditions:

```
DCL EXCM * EXCID(H' 2201' ) BP(LI BRARY-NOT-FOUND) CV(X' 00000000' );
LI BRARY-NOT-FOUND:
```

```
    CPYBLAP      S-MESSAGE, "Li brary not found...", " ";
    CPYBLA       A-LI BRARY, "e";
    B            CLEAR-OBJECT-I NFO;
```

```
OBJECT-NOT-FOUND:
```

```
    CPYBLAP      S-MESSAGE, "Ob ject not found...", " ";
    CPYBLA       A-OBJECT, "e";
    CPYBLA       A-TYPE, "e";
```

```
CLEAR-OBJECT-I NFO:
```

```
    CPYBREP      S-CREATI NG-USER, " ";
    CPYBREP      S-CREATI NG-SYSTEM, " ";
    CPYBREP      S-ALLOW-USER-CHANGE, " ";
    CPYBREP      S-CHANGED-BY-PGM, " ";
    CPYBREP      S-USER-DEFI NED-ATTR, " ";
    CPYBREP      S-PTF-LEVEL, " ";
    CPYBREP      S-VERSI ON-RELEASE, " ";
    CPYBREP      S-TEXT, " ";
    CPYBREP      S-PROD-LANGUAGE-I D, " ";
    CPYBREP      S-PROD-OPTI ON-I D, " ";
    CPYBREP      S-COMPONENT-I D, " ";
    CPYBREP      S-ATTRI BUTE, " ";
    CPYBREP      S-SAVE-TI ME, " ";
    CPYBREP      S-RESTORE-TI ME, " ";
    CPYBREP      S-CREATI ON-DATE, " ";
    B            SHOW-ERROR-MESSAGE;
```

```
SUPPLY-OBJECT:
```

```
    CPYBLAP      S-MESSAGE, "Supply Obj ect name", " ";
    CPYBLA       A-OBJECT, "e";
    B            CLEAR-OBJECT-I NFO;
```

```
SUPPLY-LI BRARY:
```

```
    CPYBLAP      S-MESSAGE, "Supply Li brary name", " ";
    CPYBLA       A-LI BRARY, "e";
    B            CLEAR-OBJECT-I NFO;
```

```
DCL EXCM * EXCID(H' 0C01' ) BP(SUPPLY-TYPE) CV(X' 00000000' );
```

```
SUPPLY-TYPE:
```

```
    CPYBLAP      S-MESSAGE, "Supply val id Obj ect type/subtype", " ";
    CPYBLA       A-TYPE, "e";
    B            CLEAR-OBJECT-I NFO;
```

```
DONE:
```

```
    CPYBLA       CTRL-OPCODE, "CLOSE";
    CALLX        .MI SCRNI O, MI SCRNI O, *;
    RTX          *;
```

We only show part of the screen description (you get the idea):

```
/***** SCREEN DEFINITION *****/
```

```
DCL SPCPTR .SCRN-CTRL I NIT(SCRN-CTRL);
```

```
DCL DD SCR N-CTRL CHAR(8);
```

```
    DCL DD CTRL-OPCODE          CHAR(1) DEF(SCRN-CTRL) POS(1);
```

```
    DCL DD CTRL-CMD-KEY         ZND(2,0) DEF(SCRN-CTRL) POS(2);
```

```
    DCL DD CTRL-CURSOR-POSITI ON CHAR(5) DEF(SCRN-CTRL) POS(4);
```

```
    DCL DD CTRL-CURSOR-ROW ZND(2,0) DEF(CTRL-CURSOR-POSITI ON) POS(1);
```

```
    DCL DD CTRL-CURSOR-COL ZND(3,0) DEF(CTRL-CURSOR-POSITI ON) POS(3);
```

```
DCL SPCPTR .SCREEN I NIT(SCREEN);
```

```
DCL DD SCREEN          CHAR(10) I NIT("T010310020");
```

```
DCL DD S-TITLE          CHAR(20) I NIT(" Work wi th OIR ");
```

```
DCL DD *                CHAR(10) I NIT("L010520000");
```

```
DCL DD *                CHAR(10) I NIT("L030010030");
```

```
DCL DD *                CHAR(30) I NIT("Ob ject name . . . . . ");
```

```
DCL DD A-OBJECT          CHAR(10) I NIT("i 030320010");
```

```
DCL DD S-OBJECT          CHAR(10) I NIT("(10)" );
```

```
DCL DD *                CHAR(10) I NIT("L030430022");
```

```
DCL DD *                CHAR(22) I NIT("          Name          ");
```

```
...
```

```
DCL DD *                CHAR(10) I NIT("L210010030");
```

```
DCL DD *                CHAR(30) I NIT("Text descri ption . . . . . ");
```

```

DCL DD A-TEXT      CHAR(10) INIT("I210320050");
DCL DD S-TEXT      CHAR(50) INIT((50)" ");
DCL DD *           CHAR(10) INIT("L220030022");
DCL DD *           CHAR(22) INIT((22)" ");

```

We changed the attribute of S-F11 with CPYBLA A-F11, "L". Remember that the *shortest* operand (the "L") determines how much is changed:

```

DCL DD A-F3  CHAR(10) INIT("U230010010");
DCL DD S-F3  CHAR(10) INIT("F3=Exit ");
DCL DD A-F11 CHAR(10) INIT("U230110010");
DCL DD S-F11 CHAR(10) INIT("F11=Change");
DCL DD A-F12 CHAR(10) INIT("U230230010");
DCL DD S-F12 CHAR(10) INIT("F12=Cancel");
DCL DD *     CHAR(10) INIT("U230340000");

DCL DD A-MESSAGE CHAR(10) INIT("T240010079"); /* MESSAGE LINE */
DCL DD S-MESSAGE CHAR(79);

DCL DD * CHAR(10) INIT(".000000000"); /* END OF SCREEN */

DCL SYSPTR .MI SCRNI O;
DCL OL MI SCRNI O(.SCRN-CTRL, .SCREEN);

```

### What is in the Additional OIR Records?

It seems that externally described files (1901), modules (0301), service programs (0203), and programs (0201) are listed in the second (and higher) OIR records. Without attempting to completely decipher the structure at this point, let's simply show a couple of examples:

Object:

```

015600 FFFFFFFF 0203C3D6 D5C4C8C4 D3D94040 ..... CONDHLDR
015610 00000002 0014D9D7 C7D3C540 40404040 ..... RPGLE

```

Second record (actually starting 8 bytes before the end of the first record):

```

000000D6 0006
D8D4 C8D9C3E5 D7D44040 405CD3C9 C2D34040 40404040 0201 QMHRVCVM *LI BL
D8E4 E2D9D1D6 C2C94040 405CD3C9 C2D34040 40404040 0201 QUSRJOBI *LI BL
C3D6 D5C4C8C4 D3D9C440 40D3E2E5 C1D3C7C1 C1D9C440 1901 CONDHLDR LSVALLGAARD
C3D6 D5C4C8C4 D3D9C440 40030002 CONDHLDR
E6F0F140 40404040 4040F2F8 C4F4F3F2 F0F5C5C4 C2F6F4 00000006 W01 28D43205EDB64
C4E4D4D4 E8404040 4040F0F0 F5F2F5F1 F4C4F4C5 F8F8F1 00000000 DUMMY 0052514D4E881
D8D9 D5E7C9C5 40404040 40D8E2E8 E2404040 40404040 0203 QRNXI E QSYS
D8D9 D5E7C9D6 40404040 40D8E2E8 E2404040 40404040 0203 QRNXI O QSYS
D8D3 C5C1E6C9 40404040 40D8E2E8 E2404040 40404040 0203 QLEAWI QSYS

```

Object:

```

000C00 FFFFFFFF 0201C7C5 E3E2C5D7 E3404040 ..... GETSEPT
000C10 00000002 0005C3D3 D7404000 00000000 ..... CLP .....

```

Second record:

```

0000009A 0004
E2C5 D7E3C4E4 D4D74040 4050D3C9 C2404040 40404040 1901 SEPTDUMP &LI B
E2C5 D7E3C4E4 D4D74040 400B0000 SEPTDUMP
D8C4 C4E2E2D9 C3404040 405CD3C9 C2D34040 40404040 1901 QDDSSRC *LI BL
D8C4 C4E2E2D9 C3404040 40010000 QDDSSRC
D8D7 E2D9E5C4 D4D74040 40404040 40404040 1901 QPSRVDMP
D8D7 E2D9E5C4 D4D74040 40030000 QPSRVDMP
E2C5 D7E3D3C9 E2E34040 4050D3C9 C2404040 40404040 1901 SEPTLI ST &LI B
E2C5 D7E3D3C9 E2E34040 40090000 SEPTLI ST

```

Blank

## Chapter 32

### Immune against Check Object Integrity

#### Checking Object Integrity

One of the weapons supplied by IBM in the arsenal to fight unauthorized access to objects on an AS/400 is the “Check Object Integrity” command (**CHKOBJI TG**). This command reports any program objects belonging to a user profile that have been “altered” (usually with SST). Such altered programs may have the system state attribute set or may be using blocked MI-instructions and are therefore a security risk. Note that I say “risk”, not “breach”. We all take risks every day, many of which are known. As long as a risk is known one can evaluate if it is worth taking (often it is, as anyone who drives to work will know). Running altered system state programs is quite all right - and at times even necessary - as long as you know what they do and why. This Chapter explores if one can trust the “Check Object Integrity” command to do its job. As the title of the Chapter suggests, all is not well.

#### Hacking **CHKOBJI TG**?

It is quite possible to “hack” the **CHKOBJI TG** command-processing program (**QSYCHKI T**) so as to bypass checking for selected programs, but this is not for the faint of heart and is therefore probably beyond most would-be attackers. In the security business the defender has by far the toughest job. The attacker does not care about how strong the defense is overall. All he needs is one weakness. Access to System Service Tools (SST) must be closely controlled. There might be a legitimate reason for allowing a programmer access to SST for a brief period. We should be able to use **CHKOBJI TG** to check that no programs have been altered during that time and left for later, unauthorized access and use. So, the question is: “is there a simple, easy to use way of ‘immunizing’ an altered program against detection by **CHKOBJI TG**?, one that the ‘script kiddies’<sup>1</sup> could use?” Unfortunately, the answer seems to be “yes”.

#### Using **CHKOBJI TG**

Let’s first see **CHKOBJI TG** in action. Compile any program, say A, and use SST to change it to have the system state attribute. Then run **CHKOBJI TG**:

```
====> CHKOBJI TG USRPRF(LSVALGAARD) OUTFILE(XXX)
Command completed successfully but violations found.
```

The altered program is detected and a record is written to the XXX outfile:

```
====> DSPPFM XXX
```

File	Display	Physical	File	Member
File . . . . .	XXX		Library . . . . .	LSVALGAARD
Member . . . . .	XXX		Record . . . . .	1
Control . . . . .			Column . . . . .	1
Find . . . . .				
* . . . . . 1 . . . . . 2 . . . . . 3 . . . . . 4 . . . . . 5 . . . . . 6 . . . . . 7 . . . . .				
1042301154756MYAS400	OA		LSVALGAARD*PGM	LSVALGAARDPGMMOD
***** END OF DATA *****				

So far, so good.

---

<sup>1</sup> ‘Script kiddies’ for the non-informed, are the lowest forms of hackers. In fact, they’re not really true “hackers”; they just take other peoples hard work in finding exploits, and use it to create an annoyance – often not even knowing what the exploit actually does (Akin to people in Sales and Marketing ☺.)

## Disowning an Object

Since CHKOBJI TG checks objects owned by a user profile, the very first thing an attacker would think of would be to somehow sever the ownership connection between the user profile and the object. So, he starts up SST and drills down to object A owned by user profile LSVALGAARD:

====> STRSST

(We omit the tedious details on how to get there. See Chapter 6 on how to use SST.)

Note that SST knows (or checks) that object "A" is owned by the user profile LSVALGAARD:

Display Object Found Information	
Output device . . . . .	Display/Alter storage
Object:	
Type . . . . .	(02) - Program
Name . . . . .	A
Subtype . . . . .	01
Context:	
Name . . . . .	LSVALGAARD
Subtype . . . . .	01
User profile:	
Name . . . . .	LSVALGAARD
Subtype . . . . .	01
Press Enter to continue.	

Di spl ay Storage									
Control . . . . .			nnnnn, Pnnnn, Lcccccc, .cccccc, >						
Address . . . . .			1EA6BCE917 000000						
0000	00010010	00900001	1EA6BCE9	17000000	*	. . . . . Z . . . *			
0010	70010300	00000000	15506B5F	F8001000	*	. . . . . &, .8. . *			
0020	80000201	C1404040	40404040	40404040	*	. . . . . A . . *			
0030	40404040	40404040	40404040	40404040	*	. . . . . *			
0040	40408000	00000000	00000030	FF1C3600	*	. . . . . *			
0050	8259CAE3	F6598000	053B6FC9	F9000000	*	. . . T6. . . ?I9. . *			
0060	00000000	00000000	01AA965C	51000000	*	. . . . . *			
0070	1EA6BCE9	17001000	10000000	01000000	*	. . . Z. . . . *			
0080	8259CAE4	38488000	00000000	00000000	*	. . . U. . . . . *			
0090	00000000	00000000	00000000	00000000	*	. . . . . *			
00A0	0000FF1C	00000032	00000000	00000000	*	. . . . . *			
00B0	00000000	00000000	00000000	00000000	*	. . . . . *			
00C0	00000000	00000000	E0010000	00000000	*	. . . . . *			
00D0	00000000	00000000	00000000	00000000	*	. . . . . *			
00E0	00000000	00000000	00000000	00000000	*	. . . . . *			
00F0	00320000	00000000	00000000	00000000	*	. . . . . *			
F3=Exit		F4=Al ter l abels		F5=Refresh		F6=Di spl ay stack			
F9=Pop stack		F10=Push stack		F11=Al ter storage		F12=Cancel			

0040	40408000 00000000	00000030 FF1C3600	* . . . . . *
0050	8259CAE3 F6598000	<b>003B6FC9 F9000000</b>	* . . T6. . . ?I9. . *
0060	00000000 00000000	01AA965C 51000000	* . . . . . *

The data has been altered. The changed data is displayed.

Having located the address of the user profile at offset x'0058', our intrepid hacker simply changed some bits of the address, thus invalidating the link to the user profile. And sure enough, now CHKOBJI TG cannot find the object and will thus not report it as altered:

====> CHKOBJI TG USRPRF(LSVALGAARD) OUTFILE(XXX)  
**Command completed successfully.**

What happens if we reinstate the ownership by changing the address back to its original value? You didn't think that CHKOBJI TG all the sudden could find the object again, did you? Because if you did, you'd be wrong. The ownership is severed permanently. An IPL does not restore the ownership.

## The User Profile as a Machine Index

Objects that you own you have full authority over. You may also have authority to other objects. All objects that you have specific authority to are listed in a machine index that resides *inside* your user profile object. The “Materialize Authorized Objects” MI-instruction (**MATAUOBJ**) can be used to retrieve all objects that you own. It is conjecture only, but I think that the instruction checks to see that the object exists and that you indeed are the owner as given in the object header. If neither is true, the object is removed from the index and the ownership is severed. This defeats CHKOBJI TG.

## Defense in Depth (The Alter Log)

Good security has “defense in depth”; *i.e.* does not rely on a single lock, barrier, or obstacle. SST tries valiantly to provide at least some auditing of what it is being used for. If our attacker tries to defeat CHKOBJI TG as outlined above using SST to sever the ownership relation, SST makes an entry in its *alter log*. To inspect the alter log, start SST and select “Display/Alter storage”, then:

Select Data	
Output device . . . . .	Display/Alter storage
Select one of the following:	
1. Machine Interface (MI) object	
2. Licensed Internal Code (LIC) data	
3. LIC module	
4. Tasks/Processes	
5. Starting address	
Selection	
2	

Select LIC Data	
Output device . . . . .	Display/Alter storage
Select one of the following:	
1. Node Address Communication Area (NACA)	
2. Machine Initialization Status Record (MISR)	
3. Source/Sink Active Device List (SSADL)	
4. Recovery list	
5. Database in use table	
6. Attached commit block table	
7. Alter Log	
8. Machine index	
9. LIC Link Map	
10. Heap space	
11. Main storage usage trace	
12. Transport manager traces	
13. Storage management functional trace	
14. Allow fix apply on altered LIC	
Selection	
7	

Specify Alter Log Entries	
Output device . . . . .	Display/Alter storage
Type choices, press Enter.	
Address range:	
From . . . . .	0000000000 000000 0000000000 000000-FFFFFFFF FFFFFF
To . . . . .	0000000000 000000 0000000000 000000-FFFFFFFF FFFFFF
Replaceable unit . . . . .	_____ Name
User profile . . . . .	_____ Name, *DST
Date range:	
From . . . . .	____/____/____ MM/DD/YY
To . . . . .	____/____/____ MM/DD/YY
Note: Entries in the Alter Log which meet all specified values will be displayed.	

Specify various ranges (note that the date range is not Y2K compliant), or just press Enter to select all entries. They will be shown one at a time. Here is the first one (all entries shown are fake and are for illustration only):

Display Alter Log Entry					
Date/Time of alter	:	06/22/00	09: 03: 42	MM/DD/YY	HH: MM: SS
User profile	:	QSECOFR			
Address altered	:	04E1D5C21B 0014D0			
Data:					
Old	:	7C0004C8	E05B000B	7C0004C8	785C17A1 @..H.\$..@..H.*..
New	:	7C0004C8	E05B000B	48000034	785C17A1 @..H.\$.....*..
Current	:	7C0004C8	E05B000B	48000034	785C17A1 @..H.\$.....*..

The log shows that user QSECOFR altered data at the shown address on 06/22/00.

## Format of Alter Log Entries

You can use SST (or similar) to actually look more closely at the Alter log (but note that SST will not allow you to *change* Alter log entries - more defense in depth). The Alter log has a fixed address (segment x'000000004E')

Address	000000004E	000000			
000000	20880800	00800000	00000000	4E000000	.h..0.....+..
000010	00010000	00000000	00000000	00000000	.....
000020	C1D3E3C5	D940D3D6	C740C8C5	C1C4C5D9	ALTER LOG HEADER
000030	F4F2F000	00000000	00000000	00000000	420.....
000040	00000000	00000000	00000000	00000000	.....
000050	7F3DD191	B8330000	00000000	00000000	"Jj ½.....+.. ← next free entry
000060	00000000	4E000020	80000000	00000000	.....+..0.....
000070	04E1D5C2	1B0014D0	0201D4C1	D2C5D7E3	..NB..}..MAKEPT ← one entry
000080	D9404040	40404040	40404040	40404040	R
000090	40404040	40404040	00000000	00000004	.....
0000A0	F0F0F0F6	F2F2F0F9	F0F3F4F2	80000000	0006220903420..
0000B0	7C0004C8	E05B000B	7C0004C8	785C17A1	@..H\\$. @..H  *..~
0000C0	7C0004C8	E05B000B	48000034	785C17A1	@..H\\$. c...  *..~
0000D0	D8E2C5C3	D6C6D940	40404040	40404040	QSECOFR
0000E0	40404040	40404040	40404040	4040FFEO	. \
0000F0	0000FF78	00FF0000	00000000	00000000	...}.....
000100	00000000	00000000	00000000	00000000	.....
000110	FFFFFFF7	C8359C30	7BD9C9C4	C9C1C7C6	...H.æ.#RI DI AGF
000120	00000000	00000000	00000000	00000000	.....
000130	00000000	00000000	00000000	0000002C	.....
000140	F0F0F1F2	F1F9F1F1	F1F6F5F9	40000000	001219111659...
000150	FFFFFFF7	C156FA90	FFFFFFF7	C43C7490	...Aî³°...D.Ë°
000160	FFFFFFF7	C156FA90	FFFFFFF7	C43C6798	...Aî³°...D.Âq
000170	C2C1C9C7	E4E84040	40404040	40404040	BADGUY
000180	40404040	40404040	40404040	4040F860	8-
000190	0000FF78	00000000	00000000	00000000	...}.....
0001A0	00000000	00000000	00000000	00000000	.....
0001B0	00000000	00000000	00000000	00000000	.....
0001C0	00000000	00000000	00000000	00000000	.....

The log entry even contains the name of the object (if known) that has been altered. This information is not available on the SST Alter log display. The addresses shown in the log are truncated down to a 16-byte aligned value. Here is the data declaration for an entry:

```

DCL DD ALTER-ENTRY CHAR(160) BDRY(16);
DCL DD THE-ADDRESS CHAR(8) DEF(ALTER-ENTRY) POS( 1);
DCL DD THE-OBJECT CHAR(32) DEF(ALTER-ENTRY) POS( 9);
DCL DD * CHAR(8) DEF(ALTER-ENTRY) POS( 41);
DCL DD YYMMDDHHMSS CHAR(12) DEF(ALTER-ENTRY) POS( 49);
DCL DD * CHAR(4) DEF(ALTER-ENTRY) POS( 61);
DCL DD CONTENTS-BEFORE CHAR(16) DEF(ALTER-ENTRY) POS( 65);
DCL DD CONTENTS-AFTER CHAR(16) DEF(ALTER-ENTRY) POS( 81);
DCL DD USER-PROFILE-NAME CHAR(30) DEF(ALTER-ENTRY) POS( 97);
DCL DD AUTHORITY-BITS CHAR(8) DEF(ALTER-ENTRY) POS(127);
DCL DD * CHAR(26) DEF(ALTER-ENTRY) POS(135);

```



### ***Clearing the Alter Log***

The address at offset x'58' points to the next free entry. If you set this value to x'000000004E 000070' (which is the address of the first entry), the Alter log will appear empty. Since you cannot use SST to change the Alter log, you must use the technique described in Chapter 7 to forge a pointer to the Alter log. Our evil attacker could do this too, of course, and even erase all of the entries, or, worse: insert fake entries to damage the reputation of co-workers, or other such mischief. The message: just because the super-secure system says so, it ain't necessarily so.

### ***How Did We Know the Address of the Alter Log?<sup>2</sup>***

On any system, low memory is a popular place for all kinds of system wide and important stuff to be located. Simply looking at segments one at a time quickly revealed the address of the Alter Log (and of many other interesting objects). One could try to give the log a less obvious name, but the attacker would surely recognize the names of the programs and modules (or even the addresses) that he has altered, so this 'defense' would not be very strong. One could try to encrypt the information in the log, but the segment would still stand out as interesting; the attacker could still simply clear it and watch what happens. And so the arms race continues.

---

<sup>2</sup> "One can observe a lot just by watching" (Yogi Berra)

Blank

## Chapter 33

### Analysis of SCV 7, Program Call

#### **Program Calls**

The target program, that you want to call, is identified by a system pointer. A program call transfers control to a pre-determined entry point within the target program called the Program Entry Point (PEP). The program call is the focal point for two important issues: the authority granted to the running process, and the program state attribute with which the process is executing are both subject to modification (by the system) at this point of control transfer.. In this Chapter we shall investigate how these attributes are managed.

#### **Adopted Authority**

The MI program object has an attribute in its header to indicate that while the program is executing (or more precisely, while it's invocation in the stack is executing), the process has available the authority of the program owner's user profile in addition to any other user profiles associated with the process. The user profile that owns the program can also be adopted for use by the process as a supplemental source of authority. The adopt-owner-authority bit is set when the program is created.

A program, by default, runs with the same 'inherited' authorities of existing programs on the stack that already adopt owners' authorities. In addition to the adopt-owner-authority bit in the program header, another bit, known as the propagate bit, is defined. This bit stops or allows the use of any adopted authority from invocations above it on the stack. The three bits, adopt-owner, propagate-adopted, and suppress-adopted are stored in the program header. When a program is activated, the routine that does the activation (the SLIC call code) sets these bits in the Invocation Control Block (ICB). During execution, each time authority is checked, the access control algorithm walks up the invocation stack, checking at each level for adopted authority or a bit indicating no propagation of adopt or a bit indicating that adopted authority should be suppressed.

#### **The Program State Attribute**

Every program has an associated execution state (user, system, or inherit). All objects, including programs, have an associated domain (user or system). A state transition can occur when the machine invokes or terminates a program. All program objects contain state bits in the program header that determine what state the process will be in when it runs the program object. The thread execution state transitions (from user to system and back) are managed by the SLIC call code and SLIC exception management.

The current execution state of the process is kept in a bit in the Machine State Register (MSR) as we saw in Chapter 11. The execution state of the previous invocation is kept in each Invocation Stack Frame (ISF) for correctly restoring the execution state as the stack is unwound. When a task is "swapped out" (enters a wait state), the contents of the MSR are saved in the Task Object. When a task is dispatched, the contents of the MSR that was saved are loaded back from the Task Object into the MSR in the processor, thus restoring the execution state (user or system) to what it was when the task was swapped.

#### **Generated Code for CALLX Instruction**

We start with a simple test program that calls another external program using the CALLX MI-instruction:

```
DCL SPCPTR . PARM1 INIT(PARM1);
DCL DD      PARM1 CHAR(10);

DCL SPCPTR . PARM2 INIT(PARM2);
DCL DD      PARM2 CHAR(10);

DCL OL ARGUMENT-LIST (. PARM1, . PARM2) ARG;
DCL SYSPTR . MYPROG;
```

```
ENTRY * EXT;
CALLX .MYPROG, ARGUMENT-LIST, *;
RTX      *;
```

The generated RISC code for the CALLX instruction looks like this:

```
0014CC 381E0060 ADDI 00 30 96 [· · -] addr of PARM2
0014D0 385E0040 ADDI 02 30 64 [· · -] addr of PARM1
0014D4 389E0080 ADDI 04 30 128 [· · 0] R4 = addr of .MYPROG
0014D8 386100B0 ADDI 03 01 176 [· / ^] R3 = addr of ARGUMENT-LIST
0014DC F80100C0 STD 00(01) 192 [8· {] 2nd parm = addr of .PARM2
0014E0 7FC4F088 TD 30 04 30 [· · ·]
0014E4 7CA52878 ANDC 05 05 05 [· · 1] R5 = NULL (i.e. 3rd parm)
0014E8 F84100B8 STD 02(01) 184 [8· ½] 1st parm = addr of .PARM1
0014EC 38000002 ADDI 00 00 2 [· · -] 2 parameters
0014F0 7800000E RLDIMI 00 00 32 00 [1 · ·]
0014F4 F80100B0 STD 00(01) 176 [8· ^] set number of parameters (2)
0014F8 419A8073 BCLA 12 26 FFFF8070 [· · 0E]
0014FC 440000E1 SCV 7 [· · ·] execute the call
001500 419A8083 BCLA 12 26 FFFF8080 [· · 0C]
```

The two **BCLA** instructions are for tracing purposes. We see the **SCV** instruction that affects the actual call. The registers R3, R4, and R5 are set up with addresses of the three operands of the **CALLX** instruction. Although the register sequence usually matches the operand sequence, that is not the case here, since R4 points to the first operand and R3 points to the second operand.

## The **SCV** 7 Dispatch Code

Back in Chapter 11, we determined that program calls are executed through the SLIC module **#AICAPGM**, but before we get to that, here is the dispatch code for SCV 7 (see Chapter 18 for more on SCV):

```
0030E0 7C0902A6 MFSPR 00 09 [· · w] R0 = count register (old MSR)
0030E4 7D8000A6 MFMSR 12 [· · w] R12 = MSR (new Machine State Register)
0030E8 780C042C RLDIMI 12 00 00 48 [1 · ·] R12[48:63] = same bits from R0 (old MSR)
0030EC 79AC6CAC RLDIMI 12 13 13 50 [· · 0D] R12[50] = 0 (disable floating point)
```

When a normal interrupt occurs there are two registers that must be saved: the current machine state and the return address. The PowerPC has special registers for that, the SSR1 and the SSR0, respectively. The SCV interrupt is special in the sense that other special registers are used instead, namely the Link Register (SR8) and the Count Register (SR9). The above code builds in R12 the value of the old (before the SCV) MSR, except that floating point operations are turned off. This somewhat tricky code was explained more fully back in Chapter 18. The value now in R12 will become the new MSR upon return to the caller. Finally we simply go to the SLIC call module:

```
0030F0 419E8862 BCA 12 30 FFFF8860 [· · hA] tracing if enabled
0030F4 3C40B161 ADDIS 02 00 -20127 [· · E/] R2 = FFFF FFFF B161 0000
0030F8 4A6D0C5E BA FE6D0C5C [· · -] go to #AICAPGM (addr FFFFFFFF FE 6D0C5C)
```

## The **#AICAPGM** Module

The address shown is for V4R4M0. It will be different on other releases, but you can use SST to find the module by name. At all times R1 points to the Invocation Stack Frame (ISF). As always, a module begins with some “housekeeping” code:

```
FFFFFFFE 6D0C5C
6D0C5C F9C1FF73 STMD 14(01) -144 [9A· E] Save R14 thru R31
6D0C60 7C0802A6 MFSPR 00 08 [· · w] get Link register with return addr
6D0C64 F8010028 STD 00(01) 40 [8· ·] save return address
6D0C68 F821FA81 STDU 01(01) -1408 [8· 3a] reserve 1408 bytes of stack space
6D0C6C 3C006010 ADDIS 00 00 24592 [· · -]
6D0C70 F8010008 STD 00(01) 8 [8· ·]
6D0C74 E8029678 LD 00(02) -27016 [Y· ol]
6D0C78 F8010010 STD 00(01) 16 [8· ·]
6D0C7C F9810020 STD 12(01) 32 [9a· ·] store return value of MSR in ISF
6D0C80 38000000 ADDI 00 00 0 [· · -]
6D0C84 F8010080 STD 00(01) 128 [8· 0]
6D0C88 7C3C0B78 OR 28 01 01 [· · l] R28 -> ISF
6D0C8C EB629670 LD 27(02) -27024 [0A00]
```

Make copies of pointers to the three operands (because the contents of R3, R4, and R5 are often destroyed):

```

6DOC90 7C721B78 OR      18 03 03    [ @É. l ] R18 = 2nd operand (parameter list)
6DOC94 7C9A2378 OR      26 04 04    [ @ª. l ] R26 = 1st operand (program to call)
6DOC98 7CBD2B78 OR      29 05 05    [ @·. l ] R29 = 3rd operand (authority)

```

When an SCV interrupt is taken, MSR[49] is cleared so that the processor runs in supervisor mode, but MSR[56] (the program state bit - system or user) is not changed. If the program state bit were changed, the following code would not make sense:

```

6DOC9C 7FC000A6 MFMSR    30          [ "{ w ] R30 = MSR
6DOCA0 41958023 BCLA     12 21 FFFF8020 [ · n0· ]
6DOCA4 3A000001 ADDI      16 00          [ · · ] R16 = 1
6DOCA8 73CC0080 ANDI      12 30 000128 [ ¨ö 0 ] if MSR[56] = 0 (system state)
6DOCAC 40820008 BC        04 02 +8      [ b · ] R16 = 128 (x'80')
6DOCB0 3A000080 ADDI      16 00 128     [ · 0 ] endi f

```

At this point **R16** contains 1 if the calling program is running in user state or 128 if the calling program is running in system state.

## Check State Attribute

We now skip down to the following code. In the code skipped, R26 was set to point to the address of the target program object. We now check if the program domain and program state attribute form an allowed combination:

```

6DOEA0 73CC0080 ANDI      12 30 000128 [ ¨ö 0 ] if MSR[56] = 1 (user state)
6DOEA4 41820018 BC        12 02 +24     [ · b · ] check domain of callee:
6DOEA8 899A0006 LBZ       12(26) 6      [ iª · ] pickup domain (at offset 6)
6DOEAC 718C0080 ANDI      12 12 000128 [ ¨ö 0 ] if domain is system
6DOEB0 4182000C BC        12 02 +12     [ · b · ] (remember state is user)
6DOEB4 7F49D378 OR        09 26 26     [ ¨ll ] error trap
6DOEB8 7FE00008 TW        31 00 00     [ "\ · ] end both ifs

```

The above code checks to see if a user state caller tries to call a program in the system domain. If so, a trap is executed. Replacing the trap instruction (**TW**) with a NOOP is a security breach; if this has been done, you know that your system has been tampered with in an interesting and maybe nefarious manner.

The state attribute of the target program (the *callee*) is stored as a halfword in the program header (as we explored in Chapter 15). Set R29 to point to the program header:

```

6DOEDC EBBA0070 LD        29(26) 112 [ 0[ 0 ] R29 -> program header

```

Then check (remember that R16 contains the value of the state attribute with which the caller is running):

```

6DOF7C A2FD005C LHZ       23(29) 92    [ sÜ * ] state to use = target.state
6DOF80 7CB78000 CMP       1L 23 16     [ @¼0 ] if target.state not = caller.state
6DOF84 41860034 BC        12 06 +52     [ · f · ] check if inherit state:
6DOF88 2CB70000 CMPI      1L 23 000000 [ ¼ ] if target.state = 0 (inherit)
6DOF8C 4086000C BC        04 06 +12     [ f · ] state to use = caller.state
6DOF90 7A170420 RLDI CL   23 16 00 48 [ : · · ]
6DOF94 48000024 B         +36 [ ç · ] else

```

If the target program has the “inherit state” attribute it will be allowed to run with the state of the caller. We now skip some more housekeeping code that retrieves the program entry point address and the address of the constants area. We are now ready to set up the environment for the program to call. System state programs and user state programs run in different activation groups. Data declared in a system state program cannot be changed by user state programs (see Chapter 15 for more on page protection):

```

6D109C 2CB70080 CMPI      1L 23 000128 [ ¼ 0 ] if state to use = system state
6D10A0 40860020 BC        04 06 +32     [ f · ] then (otherwise go around to 6D10C0)

6D10A4 E99C0290 LD        12(28) 656   [ Zª º ] Use system state:
6D10A8 E98C0020 LD        12(12) 32    [ Z0 º ] select System Activation Group
6D10AC F99C0290 STD       12(28) 656   [ 9ª º ]
6D10B0 73CC0080 ANDI      12 30 000128 [ ¨ö 0 ] R30 is still a copy of MSR
6D10B4 41820024 BC        12 02 +36     [ · b · ]
6D10B8 6BDE0080 XORI      30 30 000128 [ ¨ú 0 ] copy of MSR[56] = 0 (system state)
6D10BC 4800001C B         +28 [ ç · ] el se

6D10C0 E99C0290 LD        12(28) 656   [ Zª º ] Use User state:
6D10C4 E98C0018 LD        12(12) 24    [ Z0 º ] select User Activation Group
6D10C8 F99C0290 STD       12(28) 656   [ 9ª º ]
6D10CC 73CC0080 ANDI      12 30 000128 [ ¨ö 0 ] R30 is still a copy of MSR
6D10D0 40820008 BC        04 02 +8      [ b · ]

```

```
6D10D4 63DE0080 ORI      30 30    000128 [Äü Ø]    copy of MSR[56] = 1 (user state)
```

R30 is now set up with the correct state bit (MSR[56]). We now skip down to where R30 is used:

```
6D1224 63DE4000 ORI      30 30    016384 [Äü  ]    copy of MSR[49] = 1 (problem mode)
6D1228 7FC00164 MTMSRD  30      [ " { . Ä ]    set MSR
6D122C 4C00012C I SYNC          [ < . . ]
```

The problem mode bit is set in the MSR together with the program state bit. We pointed out in Chapter 11 that if you (or someone) change the ORI instruction to 63DE0000, the problem mode bit will *not* be set and the machine will always run in supervisor mode (at least when executing *your* code). If you are concerned about security on your system, you may want to check this.

## Finally Calling the Program

After all this (and more - as we skipped over authority) checking, we are finally ready to call the program:

```
6D1324 7E439378 OR       03 18 18    [=äl ]    R3 = R18 -> parameter list
6D1328 39600000 ADDI     11 00      0 [ . - ]    R11 = 0 (don't know what this is for)
6D132C 7C5D1378 OR       29 02 02    [ @ . l ]    R29 -> constants for SCV 7 (why?)
6D1330 E85C02A8 LD        02(28)    680 [ Y* . y ]    R2 -> constants for target program
6D1334 E99C02A0 LD        12(28)    672 [ Zæ . µ ]    R12 = target entry point address
6D1338 7D8803A6 MTSPR    08 12      [ ' h . w ]    Link reg. = entry address from R12
6D133C 4E800021 BCLRL    20 00      [ . Ø . ]    Branch to link reg. (with LINK)
6D1340
```

Note that the BCLRL instruction stores the address (...6D1340) of the next instruction in the link register, so that the return address leads back to the SLIC (and not yet to the calling program).

## Entering the Called Program

When you enter the target program, the first few instructions – in the PEP code provided by the compiler, not the ‘actual’ program code - save the registers above Register 13 that it is going to use, retrieve and store the link register (which holds the return address to the SLIC, and reserve and build a new invocation stack frame:

```
001480 FB41FF33 STMD      26(01)    -208 [ Ü . . ]    Store R26, R27, ..., R31 below current stack
001484 605C0000 ORI       28 02    000000 [ - * ]    R28 = R2 (points to program constants)
001488 7C0802A6 MFSPR     00 08      [ @ . w ]    R0 = link register (return address)
00148C F8010028 STD        00(01)     40 [ 8 . . ]    Store return address on current stack
001490 F821FE41 STDU      01(01)    -448 [ 8 Ü . ]    Reserve new stack frame for target
```

## The STDU RISC-Instruction

The “Store Double Word with Update” instruction STDU Rs, (Ra)ds stores the contents of register Rs in the double word addressed by register Ra with offset ds, then stores that address in register in Ra. All this is done indivisibly as one “atomic” action.

The net result of the particular instruction STDU 01(01) -448 is then to create a new invocation stack frame of size 448 bytes. This particular size is for this one program only; other programs would need to reserve a stack frame of a different size as fitting for their storage requirements. The first double word of the new stack frame holds the address of the previous frame and R1 points to the new frame. Since the multiple registers were saved at a *negative* offset from the contents of the previous value of R1, they end up being actually stored within the new stack frame, as they should be. The return address is stored in the previous stack frame at offset x’28’.

## Returning from the Called Program

The RTX MI-instruction generates code that discards the current stack frame (by adding the size of the current frame to the frame address), retrieves the return address, restores the saved registers, and finally returns to the SLIC:

```
001850 382101C0 ADDI      01 01     448 [ . . . { ]    revert to previous stack frame
001854 E8010028 LD         00(01)     40 [ Y . . ]    get saved link register
001858 7C0803A6 MTSPR      08 00      [ @ . w ]    Link reg. = saved value
00185C EB41FF33 LMD        26(01)    -208 [ 0 . . ]    restore saved registers
001860 4E800021 BCLRL      20 00      [ . Ø . ]    return to SLIC (with LINK)
```

Back in the SLIC code, there are many more housekeeping chores to perform. We'll not spend time on these and we jump instead directly down to where the (real) return to the original caller is managed. When the SCV 7 code was entered a *stack frame was build* for the SLIC call code. Then when the target program was entered yet another stack frame was constructed for it. At this point, we have already disposed of the stack frame for the target; we still have to dispose of the stack frame for the SLIC call code. We can do this by loading R1 with the address stored at the beginning of the current frame, LD 01(01)0. But first we must retrieve the MSR to use when returning:

```

6D22D0 E8010020 LD      00(01)      32 [Y. .] get saved MSR
6D22D4 60004030 ORI      00 00      016432 [- . .] set problem mode bit, paging bits
6D22D8 7C0903A6 MTSPR    09 00      [ @. .w] R9 now holds the MSR to be restored

6D22DC E8210000 LD      01(01)      0  [Y. .] discard SLIC stack frame
6D22E0 E8010028 LD      00(01)      40 [Y. .] get return address
6D22E4 7C0803A6 MTSPR    08 00      [ @. .w] R8 now holds return address
6D22E8 E9C1FF73 LMD      14(01)     -144 [ZA. E] restore registers
6D22EC 4C0000A4 RFSCV                    [< u] return from SCV interrupt restoring MSR

```

The cycle is now completed, we are back at the original calling program with the original machine state register contents.

### ***Format of the Invocation Stack Frame, /SF***

```

DCL DD /SF CHAR(*);
DCL DD PREV-FRAME-ADDR CHAR(8) DEF(/SF) POS( 1);
DCL DD FRAME-TYPE      CHAR(1) DEF(/SF) POS( 13);
DCL DD ADDR-OF-??      CHAR(8) DEF(/SF) POS( 17);
DCL DD MSR-TO-RESTORE  CHAR(8) DEF(/SF) POS( 33);
DCL DD RETURN-ADDRES   CHAR(8) DEF(/SF) POS( 41);

```

Blank



## Chapter 34

### Anatomy of a User Profile

#### The Central Role of a User Profile

When you sign on to an AS/400 you supply the name of a *User Profile*. The user profile is a *Machine Context* object that controls your access rights. All jobs are associated with a user profile. A user profile also serves as a repository of customizations of the system for that particular user. Most objects are *owned* by a user profile. The *authority* you have over objects and functions derives from your user profile (or the profiles of associated or supplemental *Groups*). We have already (Chapter 32) explored the list of authorized objects stored as a machine index within the user profile. In the present Chapter, we'll look at other properties of a user profile. There are very many facets of user profiles, most at higher levels of OS/400 that we'll not cover here. "User profiles" is a *big* topic.

#### Getting User Profile for a Job

Up until V5R1, a system pointer to the user profile under which a job is running was conveniently placed in the PCS:

```

Address DFD18FD3A4 000000
000000 00020008 00810001 FBD4846F 01000000 .....a...ÙMd?...
000010 40010000 00000000 DFD18FD3 A4000020 .....ÿJ±Lu...
000020 80000000 00000000 2D4E18AC E6001000 Ø.....+·ðW... Space Ptr
000030 00008000 00000000 1AA1759A A800193F ..Ø.....~îªy... Space
000040 00008000 00000000 FA7849AF 280019FF ..Ø.....³îñ®... Space
000050 00008000 00000000 0A2F0222 F60010FF ..Ø.....6... Dev Descr
000060 00008000 00000000 EEED5978 3A0004FF ..Ø.....Óßi... Library
000070 80000000 00000000 3E6F3749 7B000190 Ø.....?·ñ#...° Space Ptr
000080 80000000 00000000 DC40B665 FC000100 Ø.....ü ¶AÜ... Space Ptr
000090 00000000 00000000 22AC74B0 6F000800 .....ðÈ^?... User Prf ← thru V4R5
0000A0 00008000 00000000 E2062E81 73000AFF ..Ø.....S...aË... Queue
0000B0 00000000 00000000 00000000 00000000 .....
```

However, in V5R1 - and presumably onwards, that entry contains a NULL pointer. This is a pity, as it forces us to use a less direct and slower way of getting the same information. The Materialize Process Attributes MI-instruction, **MATPRATR**, can be used:

```

DCL SPCPTR .PROC-ATTR INIT(PROC-ATTR);
DCL DD      PROC-ATTR CHAR(32) BDRY(16);
DCL DD BYTES-PROVIDED BIN(4) DEF(PROC-ATTR) POS( 1) INIT(32);
DCL DD BYTES-AVAILABLE BIN(4) DEF(PROC-ATTR) POS( 5);
DCL DD *      CHAR(8) DEF(PROC-ATTR) POS( 9);
DCL SYSPTR  .USRPRF DEF(PROC-ATTR) POS(17); ← user profile sysptr

GET-USER-PROFILE:
MATPRATR .PROC-ATTR, *, X'16'; /* Get User Profile */
```

The second operand is an asterisk specifying the current process. The result contains a system pointer to the user profile. We can use our trusted pointer technique of chapter 7 to convert a system pointer to a space pointer. This gives us access to the functional part of the user profile.

The functional part (as usual) consists of the standard object headers followed by the object specific header. Here is the beginning of the functional part of my user profile:

```

Address 22AC74B06F 000000
000000 00010060 00918000 22AC74B0 6F000000 .....jØ...ðÈ^?...
000010 40010000 00000000 0A3A5982 BA000020 .....ßb[... ← associated space
000020 80000801 D3E2E5C1 D3C7C1C1 D9C44040 Ø...LSVALGAARD ← type/subtype = 0801
000030 40404040 40404040 40404040 40404040
000040 40408040 00000FE0 00000098 00400301 Ø ... \ ... q ...
000050 82F93DE8 06DD0000 2DA7EA09 D5000000 b9·Y·ù...x²·N... ← address of owner
...
```

## Object Specific Header

The object specific header begins at offset x '100':

```

000100  E8600000 FF780000  FFFFFFFF 00003904  8---İ.....  ← Priv-instr; Spec.auth.
000110  31C83E55 050052A0  00000001 00000280  ·H·í·êµ.....  ← Last Object Address
000120  00000000 00000001  000A04CC 00000000  .....ö.....
000130  000004CA 00000003  00000002 0000005A  .....!

...
0001C0  00000000 00000000  00000000 00000000  .....
0001D0  00080000 0000000D  DF8D51AC 50010000  .....ÿÿéb&...
0001E0  E9B5BE7E 33000000  000002D2 00000000  Z$´=.....K....
0001F0  80000000 00000000  00000000 00000000  Ø..... Audit levels

DCL DD OBJ-SPEC-HDR      CHAR(256) DEF(USRPRF-FUNCTIONAL-PART) POS(256);
DCL DD PRIV-INSTRUCTION-BITS CHAR(4) DEF(OBJ-SPEC-HDR) POS( 1);
DCL DD SPECIAL-AUTHORITY-BITS CHAR(4) DEF(OBJ-SPEC-HDR) POS( 5);
DCL DD LAST-OBJECT-ADDRESS CHAR(8) DEF(OBJ-SPEC-HDR) POS( 17);
DCL DD OBJECT-AUDIT-LEVEL CHAR(1) DEF(OBJ-SPEC-HDR) POS(248);
DCL DD USER-AUDIT-LEVELS CHAR(8) DEF(OBJ-SPEC-HDR) POS(249);

```

## Privileged Instruction Bits in User Profile

The most interesting fields of the object specific header are two bit words. The **first** word (at offset x'000' within the header - which is at x'100' from the start of the segment) contains bits governing which type of privileged instructions this user is allowed to execute; a bit is on if the instruction is allowed:

Bit Number	Privileged Function	QSECOFR	User
0	Create Logical Unit Description	1	1
1	Create Network Description	1	1
2	Create Controller Description	1	1
3	Create User Profile *SECADM	1	0
4	Modify User Profile *SECADM	1	0
5	Execute <b>DIAG</b> nose MI-Instruction	1	0
6	Terminate Machine Processing	1	0
7	Initiate Process	1	0
8	Modify Resource Management Controls	1	0
9	Create Mode Description	1	1
10	Create Class of Service Description	1	1
11 - 31	Reserved	0...0	0...0

Also shown are the bit settings for the QSECOFR user profile and for a “normal” user profile with no special privileges. Note, that the \*SECADM special authority is really bits 3 and 4 of the privileged instructions word, allowing the user profile to create and modify user profiles.

## Special Authority Bits in User Profile

The **second** word (at offset x'004' within the header) contains bit governing which *special authorities* the user profile has:

Bit Number	Special Authority	Authority	QSECOFR	User
0	All Object Authority	*ALLOBJ	1	0
1	Load (unrestricted)	*SAVSYS	1	0
2	Dump (unrestricted)	*SAVSYS	1	0
3	Suspend Object (unrestricted)	*SAVSYS	1	0
4	Load (restricted)	*JOBCTL	1	1
5	Dump (restricted)	*JOBCTL	1	1
6	Suspend object (restricted)	*JOBCTL	1	1
7	Process Control	*JOBCTL	1	0
8	Reserved		0	0
9	Service Authority	*SERVICE	1	0
10	Auditor Authority	*AUDIT	1	0

11	Spool Control	*SPLCTL	1	0
12	I/O System Configuration	*IOSYSCFG	1	0
13 - 23	Reserved		0...0	0...0
24	Modify Machine Attributes (group 2)		1	0
25	Modify Machine Attributes (group 3)		1	0
26	Modify Machine Attributes (group 4)		1	0
27	Modify Machine Attributes (group 5)		1	0
28	Modify Machine Attributes (group 6)		1	0
29	Modify Machine Attributes (group 7)		1	0
30	Modify Machine Attributes (group 8)		1	0
31	Modify Machine Attributes (group 9)		1	0

The low-order 8 bits control if the user profile is allowed to change machine attributes. Only the QSECOFR user profile has these bits on. Even if you create a user profile with a user class of security officer, **\*SECOFR**, this user will *not* get the ability to modify any machine attributes (except group 1 attributes, that are not restricted anyway).

### Object Audit Level

The **CHGUSRAUD** (Change User Audit) command allows a user with \*AUDIT special authority to set up or change auditing for a specific user if the OBJAUD keyword specifies \*USRPRF. The system value **QAUDCTL** controls turning auditing on or off.

You can audit change and read accesses by a user. Bits in the OBJECT-AUDIT-LEVEL field at offset x'0F7' determine this as follows:

Bit Number	Object Audit Level effect
0 - 5	Reserved
6	Audit object changes for this user
7	Audit object reads for this user

### User Audit Levels

The **CHGUSRAUD** (Change User Audit) command also allows a user with \*AUDIT special authority to set or change the level of activity that is audited for this user. The AUDLVL keyword field determines which levels are included in *addition* to the levels given by the system value QAUDLVL.

Bits in the USER-AUDIT-LEVELS field at offset x'0F8' determine this as follows:

Bit Number	User Audit Action for this User	AUDLVL value
0	Object deleted	*DELETE
1		
2	Security changes	*SECURITY
3	Save and Restore	*SAVRST
4		
5	Object created	*CREATE
6	Object management, such as Move and Rename	*OBJMGT
7	Job start, stop, hold, release, continue, change, disconnect, end, abnormal end, program start request (PSR)	*JOBDDTA
8	Office services, changes to system directory, use OfficeVision	*OFCSRVR
9		
10	Use of System Service Tools (SST)	*SERVICE
11	Spool file operations	*SPLFDDTA
12	System management functions	*SYSMGT
13	Use of Optical devices	*OPTICAL
14 - 15		
16	CL commands, S/36 control commands	*CMD

17 - 31		
32	Authority obtained through program adoption	*PGMADP
33 - 63		

## A Dangerous Program

A program that manufactures a pointer to the object specific header can readily change the authority bits for any user profile with the effect that that user obtains full access to your machine. This change of authority cannot be audited. The program would run in system state to be able to access the header. A particularly dangerous variation of the program saves the current settings of the authority bits, changes the bits to all ones, then shows a command line (e.g. using SEPT(1852) - **QUSCMDLN**). When the command line processing program returns, the authority bits are changed back to their regular settings.

## The Associated Space

The primary associated space holds additional information used by OS/400 to start jobs running under the profile. Here we identify some of the more important pieces of information:

```
DCL SPCPTR .UPRFSPC.
DCL DD UPRFSPC CHAR(4096) BAS(.UPRFSPC);
DCL DD UPRF-INITIAL-PROGRAM CHAR(20) DEF(UPRFSPC) POS( 17);
DCL DD UPRF-JOB-DESCR CHAR(20) DEF(UPRFSPC) POS( 37);
DCL DD UPRF-MSG-QUEUE CHAR(20) DEF(UPRFSPC) POS( 73);
DCL DD UPRF-OUTPUT-QUEUE CHAR(20) DEF(UPRFSPC) POS( 93);
DCL DD UPRF-LIMIT-CAPABILITIES CHAR(1) DEF(UPRFSPC) POS(114);
DCL DD UPRF-INITIAL-MENU CHAR(20) DEF(UPRFSPC) POS(115);
DCL DD UPRF-CUR-LIBRARY CHAR(10) DEF(UPRFSPC) POS(135);
DCL DD UPRF-SPECIAL-ENV CHAR(1) DEF(UPRFSPC) POS(145);
DCL DD UPRF-MSG-DELIVERY CHAR(1) DEF(UPRFSPC) POS(146);
DCL DD UPRF-SEVERITY-CODE-FILTER BIN(2) DEF(UPRFSPC) POS(147);
DCL DD UPRF-PRINT-DEVICE CHAR(10) DEF(UPRFSPC) POS(149);
DCL DD UPRF-ATTENTION-PROGRAM CHAR(20) DEF(UPRFSPC) POS(159);
DCL DD UPRF-USER-OPTIONS CHAR(1) DEF(UPRFSPC) POS(179);
DCL DD UPRF-LIMIT-DEVICE-SESSIONS CHAR(1) DEF(UPRFSPC) POS(191);
DCL DD UPRF-DISPLAY-SIGNON-INFO CHAR(1) DEF(UPRFSPC) POS(192);
DCL DD UPRF-ENABLED-DISABLED CHAR(1) DEF(UPRFSPC) POS(197);
DCL DD UPRF-KEYBOARD-BUFFERING CHAR(1) DEF(UPRFSPC) POS(198);
DCL DD UPRF-ASSISTANCE-LEVEL CHAR(1) DEF(UPRFSPC) POS(199);
DCL DD UPRF-CCSID BIN(4) DEF(UPRFSPC) POS(205);
DCL DD UPRF-SORT-SEQUENCE CHAR(20) DEF(UPRFSPC) POS(209);
DCL SYSPTR UPRF-QCASQ DEF(UPRFSPC) POS(241);
DCL DD UPRF-CHAR-ID-CONTROL CHAR(1) DEF(UPRFSPC) POS(739);
DCL DD UPRF-COUNTRY-ID CHAR(2) DEF(UPRFSPC) POS(1645);
DCL DD UPRF-LANGUAGE-ID CHAR(3) DEF(UPRFSPC) POS(1647);
```

The CHAR(20) names are *qualified* names (where the last 10 characters denote the library). Note, that this convention is the opposite of the way a qualified name is used in a command (library/name). A dump of the beginning of the space looks something like this:

```
Address 0A3A5982BA 000020
000020 40404040 40404040 40404040 40404040
000030 40404040 40404040 40404040 40404040
000040 40404040 F3D8C4C6 E3D1D6C2 C44040D8
000050 C7D7D340 40404040 40404040 40404040
000060 40404040 40404040 D3E2E5C1 D3C7C1C1
000070 D9C4D8E4 E2D9E2E8 E2404040 40404040
000080 40404040 40404040 40404040 40404040
000090 4040D4C1 C9D54040 40404040 5CD3C9C2
0000A0 D3404040 4040D3E2 E5C1D3C7 C1C1D9C4
0000B0 D5C20000 40404040 40404040 40405CE2
0000C0 E8E2E5C1 D3404040 40404040 40404040
0000D0 40404040 4040832C C93CC917 000040E8
0000E0 00000000 40404040 40404040 FFFFFFFF
0000F0 5CE2E8E2 E5C1D340 40404040 40404040
000100 40404040 40404040 40404040 40404040
000110 00008000 00000000 EE1EFBD9 D1000AFF
000120 40404040 40404040 40404040 40404040

          3QDFTJOB D Q ← initial program
          LSVALGAA ← job descr. to use
          RDQUSRSYS ← message queue
          MAIN *LIB ← initial menu
          L LSVALGAARD ← current library
          NB.. *S ← spec env.
          YSVAL ← attention pgm
          C.I.I... Y ← dsp signon
          .... ...U ← CCSID
          *SYSVAL ← sort sequence
          ..Ø.....Ô.ÛRJ... Queue
```

## Qualified Values

The *initial program* specifies, for an interactive job, the name of the program to be called whenever a new routing step is started that has **QCMD** as the request-processing program. If no initial program is specified, this field contains spaces. The attention program specifies the attention key handling program for this user.

The *job description* specifies the name of the job description that is used for jobs that start through subsystem workstation entries.

Further: The name of the message queue specifies the *message queue* that is used by this user. The name of the output queue specifies the *output queue* that is used by this user. The name of the initial menu specifies the *menu* that is shown when the user signs on. The name of the current library specifies the *current library* (if any - otherwise the field contains spaces) after the user signs on. The name of the sort sequence specifies the *sort sequence table* to be used for string comparisons for this user profile.

## Various User Profile Flags

There are a variety of flags that govern the capabilities, status, and the like for the user:

Flag	Value	Meaning
Assistance level:	Space	*SYSVAL
	A	*ADVANCED
	B	*BASIC
	I	*INTERMED
Special Environment:	Space	*SYSVAL
	N	*NONE
	3	*S36
Limit Device Sessions:	Space	*SYSVAL
	Y	*YES
	N	*NO
Display Signon Information:	Space	*SYSVA
	Y	*YES
	N	*NO
Limit Capabilities:	Space	*NO
	Y	*YES
	P	*PARTIAL
Message Delivery:	Space	*NOTIFY
	D	*DFT
	B	*BASIC
	H	*HOLD
Keyboard Buffering:	Space	*SYSVAL
	0	*NO
	1	*YES
	2	*TYPEAHEAD
Enabled/Disabled:	Space	*ENABLED
	D	*DISABLED

## Language, Country, and Character Set IDs

These are somewhat of a mess and need not be mutually consistent (maybe there are situations where you don't want them to be consistent). The CCSID can be a number or \*HEX (same as 65535) or \*SYSVAL (represented as -2). Different applications may treat these values (and the sort sequence) as they see fit. The MI-instructions (e.g. **CMPBLA**) do not check the various character set indications.

## User Options

This field controls various default settings such as to show detailed information or to reverse the action of the page up and page down keys (make them *roll* down or up). It is a bit field and the various bits can be combined:

Bit Number	Option	Explanation
0	*CLKWD	Keywords are shown for Commands
1	1	Always on
2	*EXPERT	Advanced information shown
3	*ROLLKEY	Reverse action of page/roll keys
4	*NOSTMSG	Status messages sent to the user are not shown
5	1	Always on
6	*STMSG	Status messages sent to the user are shown
7	*HLPFULL	Help information shown on a full screen
8	*PRTMSG	Message sent to owner when printing is complete
9	1	Always on
10	0	
11	0	
12	0	
13	1	Always on
14	0	
15	0	

## Objects Owned by User Profile

The object specific header holds the address of the last object that this user has created:

000110 **31c83e55 050052a0** 00000001 00000280 ·H·í··êµ·····0 ◀ Last Object Address

Each object is allocated a 16-byte slot; here are some of the slots before the last one used:

```

Address 31c83e5505 005200
005200 012DA6E5 A57F0019 52000000 00005200 ··wVv"··ê····ê·
005210 012528F8 0A560002 01000000 00005210 ···8·í······ê·
005220 010F4008 A683001E 01000000 00005220 ·· ·wC······ê·
005230 01084399 31760002 01000000 00005230 ··är·Î······ê·
005240 011837C6 C4020002 01000000 00005240 ··FD······ê·
005250 012BBCCD C4360002 01000000 00005250 ··òD······ê&
005260 040C28FF BB310002 01000000 00005260 ····]······ê-
005270 0114E76D D353000B 90000000 00005270 ··x_Lê··°····êø
005280 01240337 77E9000D 50000000 00005280 ···îZ···&····êø
005290 01080042 DC8C0002 01000000 00005290 ··âüð······ê°
0052a0 01071dd8 9f090002 01000000 000052a0 ···Qª······êµ
0052B0 00000000 00000000 00000000 00000000 ··········
0052C0 00000000 00000000 00000000 00000000 ··········
...
```

The slot contains a one-byte slot type followed by a ‘system pointer’ to the object. The reason for the quotes is that the pointer is not 16-bit aligned and is not tagged, so it is not really a pointer, but the 16 bytes contain the same bits as the system pointer. The last object created in our example above is identified by the system ‘pointer’ x**071dd89f09 000201**. This ‘pointer’ is often referred to as consisting of a *Segment ID Group* (the first 12 bytes) and an object type. The slot type (x’**01**’ in our example) takes on one of the following values:

- 01 Existing object (owned)
- 02 Authorized object (not owned)
- 03 Authorized user
- 04 Deleted object

Note that deleted objects are not removed from the list, but are simply marked as deleted (type 04).

## Chapter 38

### Source and Debug Information

#### ILE Programs and Modules

ILE programs (\*PGM and \*SRVPGM) can consist of many (sometimes *very* many) separately compiled *modules* (\*MODULE). Each module has its own source (member, file, and library), so the information kept in the OIR (see Chapter 31) about when and where the program was compiled and from what, cannot accommodate all that information for each module separately. In fact, the source information in the OIR is *blank* for ILE programs. If you want to debug a module, where does the debugger find the source for that module at the present time? (the source may even have been “promoted” to production since the module was compiled). For the investigation that we are going to perform in this chapter, we’ll create an ILE C-program, **I LEMAI N**, consisting of three modules: I LEMAI N, I LEMOD1, and I LEMOD2. We bind the modules together into the program with:

```
PGM
  CRTCMOD MODULE(I LEMOD1) SRCFILE(QCSRC) OUTPUT(*PRINT) +
    OPTI ON(*NOSHOWI NC) DBGVI EW(*SOURCE)
  CRTCMOD MODULE(I LEMOD2) SRCFILE(QCSRC) OUTPUT(*PRINT) +
    OPTI ON(*NOSHOWI NC) DBGVI EW(*SOURCE)
  CRTCMOD MODULE(I LEMAI N) SRCFILE(QCSRC) OUTPUT(*PRINT) +
    OPTI ON(*NOSHOWI NC) DBGVI EW(*SOURCE)
  CRTPGM  PGM(I LEMAI N) MODULE(*CURLI B/I LEMAI N +
    *CURLI B/I LEMOD1 +
    *CURLI B/I LEMOD2) +
    ACTGRP(*CALLER)
  CHGPGM  PGM(I LEMAI N) OPTI MI ZE(*FULL)
ENDPGM
```

Note that we specified a debug view of the source. The main module doesn’t do anything except call the two subordinate modules, each of which in turn just returns.

#### OIR-Information for Modules

The **MI SRCOI R** program is a modified version of the program we developed in Chapter 31 to show part of the Object Information Repository. The program displays the source library/file/member information, and also allows you to modify that information:

Work wi th OIR		
Object name . . . . .	<u>I LEMAI N</u>	Name
Library . . . . .	<u>LSVALGAARD</u>	Name
Object type/subtype . . . . .	<u>0301</u>	Hexadecimal values
Creating user . . . . .	<u>LSVALGAARD</u>	User profile
Creating system . . . . .	<u>MYAS400</u>	System name
Allow changes . . . . .	<u>-</u>	N=No, blank=Yes
Changed by program . . . . .	<u>-</u>	blank=No, Y=Yes
User defined attribute . . . . .	<u>                    </u>	Characters
PTF level . . . . .	<u>                    </u>	PP12345
Version/Release . . . . .	<u>V4R4M0</u>	VxRxMy
Source Library . . . . .	<u>LSVALGAARD</u>	Library name
Source File . . . . .	<u>QCSRC</u>	File name
Source Member . . . . .	<u>I LEMAI N</u>	Member name
Attribute . . . . .	<u>CLE</u>	Characters
Last save date/time . . . . .	<u>0000000000000000</u>	Hex timestamp
Last restore date/time . . . . .	<u>0000000000000000</u>	Hex timestamp
Creation date/time . . . . .	<u>20010725160617</u>	Date/time
Text description . . . . .	<u>Test call of I LEMOD1 and I LEMOD2</u>	
F3=Exit F11=Change F12=Cancel		

If you move (or copy) the source member to another file in another library and change the OIR information, the debugger still tries to find the original source, i.e. the debugger does not use the OIR-information.



## Change Licensed Object Description, *QLI COBJD*

Whenever possible, one should use documented interfaces, so the question arises: can we use **QLI COBJD** to change the source information in such a way that the debugger can find the new place? Here is a simple attempt, **MI LI COBJ**:

```
DCL SYSPTR .SEPT(6440) BAS(@SEPT);
DCL SPCPTR @SEPT BASPCO;

DCL SPCPTR .RETURN-LIB INIT(RETURN-LIB);
DCL DD      RETURN-LIB CHAR(10);

DCL SPCPTR .QUAL-OBJ-LIB INIT(QUAL-OBJ-LIB);
DCL DD      QUAL-OBJ-LIB CHAR(20);
DCL DD      QUAL-OBJ-NAME CHAR(10) DEF(QUAL-OBJ-LIB) POS( 1);
DCL DD      QUAL-LIB-NAME CHAR(10) DEF(QUAL-OBJ-LIB) POS(11);

DCL SPCPTR .OBJ-TYPE INIT(OBJ-TYPE);
DCL DD      OBJ-TYPE CHAR(10);

DCL SPCPTR .ERROR-CODE INIT(ERROR-CODE);
DCL DD      ERROR-CODE BIN(4) INIT(0);

DCL SPCPTR .OBJ-INFO INIT(OBJ-INFO);
DCL DD      OBJ-INFO CHAR(42);
DCL DD      INFO-NBR-RECS      BIN(4) DEF(OBJ-INFO) POS( 1);
DCL DD      INFO-KEY           BIN(4) DEF(OBJ-INFO) POS( 5);
DCL DD      INFO-LENGTH       BIN(4) DEF(OBJ-INFO) POS( 9);
DCL DD      INFO-DATA         CHAR(30) DEF(OBJ-INFO) POS(13);
DCL DD      INFO-FILE         CHAR(10) DEF(INFO-DATA) POS( 1);
DCL DD      INFO-LIB          CHAR(10) DEF(INFO-DATA) POS(11);
DCL DD      INFO-MBR          CHAR(10) DEF(INFO-DATA) POS(21);

DCL OL QLI COBJD(.RETURN-LIB, .QUAL-OBJ-LIB, .OBJ-TYPE,
                 .OBJ-INFO, .ERROR-CODE);

ENTRY * EXT;
CPYBLAP      QUAL-OBJ-NAME, "ILEMAIN", " ";
CPYBLAP      QUAL-LIB-NAME,  "*CURLIB", " ";
CPYBLAP      OBJ-TYPE,      "MODULE", " ";
CPYNV        INFO-NBR-RECS, 1;
CPYNV        INFO-KEY,     1; /* SOURCE INFO */
CPYNV        INFO-LENGTH, 30; /* FILE+LIB+MBR */
CPYBLAP      INFO-LIB, "LSVALGAARX", " "; /* try to change library */
CALLX        .SEPT(4553), QLI COBJD;
RTX          *;
```

While this *does* change the OIR-information, the debugger *still* goes after the original source. Now, this may not be too surprising as **QLI COBJD** clearly was designed (as the name implies) to change object information for *licensed* objects. Such objects are rarely (and IBM's never) shipped with debug information anyway.

## Anatomy of a Module

Here is the start of the **ILEMAIN** module just after compilation: (you can use our **MI EXPLR** tool or SST):

Address	2976D9EE8B 000000		
000000	00010030 00908000	2976D9EE 8B000000	.....°Ø..îR0»...
000010	F0010300 00000000	1104BEC0 D6001000	0.....{0... ← Associated space
000020	80000301 C9D3C5D4	C1C9D540 40404040	Ø... ILEMAIN
000030	40404040 40404040	40404040 40404040	
000040	40404000 00001000	00000060 3F103600	.....
000050	82CBD3C0 974D8000	053B6FC9 F9000000	bôL{p(Ø..?I9..
000060	00000000 00000000	01AA965C 51000000	.....j o*é...
000070	2976D9EE 8B001000	10020000 01000000	.îR0».....

The associated space address points to the data portion of the space (the only data you can have access to using straight MI). The SETSPFP instruction will give you a pointer to this data space. Judging from the contents of the space, the information here seems to begin with a *relic* from **V2R3**:

Address	1104BEC0D6 001000		
001020	00010002 00000070	00000000 00000130	..... ← size of relic
001030	C4C5C2E4 C740C4C1	E3C140C4 C5E2C3D9	DEBUG DATA DESCR
001040	5CE5F0F2 D9F0F3D4	F0F04040 40404040	*V02R03M00



```

001050 40404040 40404040 40404040 40404040
001060 40404040 00000000 00000000 00000000
001070 F1F0F1F0 F7F2F3F0 F8F1F4F2 F7000000 1010723081427...
001080 D4E8C1D3 F4F0F040 00000000 00000020 MYAS400 .....
001090 5CD4D6C4 00000001 00000020 00001BE0 *MOD- ..... \
0010A0 00000000 00000000 00000000 00000000 .....
0010B0 5CD4D6C4 00000001 00001C00 000004B0 *MOD- ..... ^
0010C0 00000002 00000000 00000000 00000000 .....

```

followed by a list of *service programs* used by the module. The list begins at offset x'1000' + the size of the relic (x'0130'):

```

001130 00000070 00000003 00000000 00000000 ...0..... ← nbr of service pgms
001140 1937D8C3 F2D3C540 40404040 D8E2E8E2 ..QC2LE QSYS
001150 40404040 40400000 00000000 00000000 .....
001160 1937D8C9 D3C54040 40404040 5CD3C9C2 ..QILE *LIB
001170 D3404040 40400000 00000000 00000000 L .....
001180 1937D8E4 E2C1D7C9 C2C44040 5CD3C9C2 ..QUSAPI BD *LIB
001190 D3404040 40400000 00000000 00000000 L .....
0011A0 00000000 00000000 00000000 00000000 .....

```

None of this is terribly useful. So, where is the information we are seeking? If we look at the *functional* part of the space (that we normally have no access to) we see this (at offset 000000):

```

Address 1104BEC0D6 00000000
000000 00020010 00818000 2976D9EE 8B000000 ..... a0..TR0»...
000010 D0010000 00000000 1104BEC0 D6001000 }..... {0... ← back address to space
000020 3892A393 21001000 00000000 00000000 .ktl..... ← HLL Symbol Table addr
000030 00000000 FF000000 00000000 00000000 .....

```

## The HLL Symbol Table

At offset x'20', there is an address of the associated space of an unnamed object:

```

Address 3892A39321 00000000
000000 00190020 00818000 2976D9EE 8B000000 ..... a0..TR0»...
000010 D0010000 00000000 3892A393 21001000 }..... ktl..... ← its associated space
000020 80000000 30001000 00000001 00000000 0.....

```

The associated space contains the *HLL Symbol Table* and other information after that:

```

Address 3892A39321 00010000
001000 00000000 00000000 00000000 00000000 .....
001010 00000000 00000000 00000000 00000000 .....
001020 C8D3D340 E2A89482 969340E3 81829385 HLL Symbol Table ← eye catcher
001030 00000000 00000035 00001BE0 00000040 ← size of symbol table
001040 00000222 00000D20 00000054 00000270 .....
001050 00000AB0 00000000 00000000 00000000 .....
001060 A3818993 00F000F1 F9F100F2 F700F1F1 tail: 0: 191: 27: 11
001070 00F6F000 C6C9D3C5 006D6D99 85A28599 .60: FILE: __reser
001080 A58584F1 006D6D82 A4869385 95006D6D ved1: __bufLen: __

```

## View Descriptors

The BIN(4) value at offset x'1038' gives the size of the symbol table, so the next piece of information is at x'1020' + x'1BE0' = x'2C00':

```

002C00 00000000 E7D7C640 E3C500F4 24000000 .... XPF TE: 4... ← XPF piece
002C10 00000000 00000000 00000000 00000000 .....

```

There is first a *space header*:

```

002C20 E5C9C5E6 40E2D7C1 C3C540C8 C5C1C440 VIEW SPACE HEAD ← eye catcher
002C30 000004A8 00000001 00000280 00000002 ...y.....0... ← total size
002C40 00000002 00000002 00000001 00000140 .....
002C50 00000280 000003A0 000003E0 00000450 ...0...μ... \... &
002C60 00000110 000003E0 FFFFFFFF 000003A0 ... \... μ ← offset to next piece
002C70 5CD5D6D5 C5404040 40404040 40404040 *NONE (all from 002C00)
002C80 40404040 40404040 40404040 40405CD5 *N
002C90 D6D5C540 40404040 40404040 40404040 ONE
002CA0 40404040 40404040 40404040 5CD5D640 *NO
002CB0 40404040 4040C3E4 C2C560F3 40404040 CUBE-3
002CC0 40404040 40404040 40400000 00000000 .....
002CD0 F1F0F1F0 F7F2F3F0 F8F1F4F2 F7000000 1010723081427... ← timestamp
002CE0 00000000 00000000 00000000 00000000 .....

```

Following the initial header, we find a number of *descriptors* with a common format exemplified by this *dummy* record:

```

002D10 C4F4D4D4 F840D9C5 C3D6D9C4 40404040 DUMMY RECORD ← eye catcher
002D20 01000000 00000030 00000000 00000000 ..... ← type & size of descr.
002D30 00000140 FFFFFFFF 6E6E6E6E 6E6E6E6E ... >>>>>>>> ← offset to next descr.

DCL DD DESCR-EYE-CATCHER CHAR(16);
DCL DD DESCR-TYPE CHAR(1);
DCL DD * CHAR(3);
DCL DD DESCR-SI ZE BIN(4);
DCL DD * BIN(4);
DCL DD * BIN(4);
DCL DD DESCR-OFFSET BIN(4); /* offset from XPF start */
DCL DD DESCR-LAST BIN(4); /* -1 if last descriptor of given type */
DCL DD DESCR-DATA CHAR(nn); /* variable length descriptor data */

```

Here is the next descriptor (a *View Descriptor*):

```

002D40 E5C9C5E6 40C4C5E2 C3D9C9D7 E3D6D940 VIEW DESCRIPTOR ← eye catcher
002D50 02404040 000000A0 00000000 00000000 ..... ← type & size of descr.
002D60 000001E0 00000280 C9D3C540 C3409996 ... \... FILE C ro
002D70 96A340A2 96A49983 8540A589 85A64040 ot source view
002D80 40404040 40404040 40404040 40404040
002D90 40404040 40404040 4040C9D3 C540C340 ILE C
002DA0 86969940 C1E261F4 F0F04040 40400000 for AS/400 ..
002DB0 00000025 01000000 0000000E F1F0F1F0 ..... 1010
002DC0 F7F2F3F0 F8F1F4F2 F5000000 FFFFFFFF 723081425 .....
002DD0 000001E0 00000230 FFFFFFFF FFFFFFFF ... \.....

```

This is clearly a *source view*. The view descriptor is followed by one or more *File Descriptors*. It is not clear why there are at times more than one file descriptor. At any rate, the *last* one of these (with DESCR-LAST = -1 or x'FFFFFFF') contains the source information we are seeking:

```

002DE0 C6C9D3C5 40C4C5E2 C3D9C9D7 E3D6D940 FILE DESCRIPTOR
002DF0 03000000 00000050 00000001 0000001E ..... & .....
002E00 00000230 FFFFFFFF 00000140 01000000 ..... ← last
002E10 D8C3E2D9 C3404040 4040D3E2 E5C1D3C7 QCSRC LSVALG ← file, lib-
002E20 C1C1D9C4 C9D3C5D4 C1C9D540 40406E6E AARDI LEMAIN >> rary, mbr

```

We have now located the File, Library, and Member of the source. It is *this* information the debugger uses to try to find the source. It is therefore this same information that we must modify to promote the module such that it can still be debugged.

For completeness, we show here the remaining descriptors:

```

002E30 E3C5E7E3 40C4C5E2 C3D9C9D7 E3D6D940 TEXT DESCRIPTOR
002E40 04404040 00000050 00000001 00000014 ..... & .....
002E50 00000280 FFFFFFFF 00000140 016B96B0 ... 0..... , o^ ← ?
002E60 01000000 00000140 00000210 0000000E .....
002E70 00000001 6E6E6E6E 6E6E6E6E 6E6E6E6E ... >>>>>>>>>>

```

Here starts a *new* View Descriptor (the *Statement View*):

```

002E80 E5C9C5E6 40C4C5E2 C3D9C9D7 E3D6D940 VIEW DESCRIPTOR
002E90 02000000 000000A0 00000000 00000000 ..... μ .....
002EA0 00000320 FFFFFFFF C8D3D340 E2A381A3 ..... HLL Stat
002EB0 85948595 A340E589 85A64040 40404040 ement View
002EC0 40404040 40404040 40404040 40404040
002ED0 40404040 40404040 4040C3E4 C2C560F3 CUBE-3
002EE0 40404040 40404040 40404040 40400153 . è
002EF0 00000000 03000001 00000006 F1F0F1F0 ..... 1010
002F00 F7F2F3F0 F8F1F4F2 F7000000 FFFFFFFF 723081427 .....
002F10 FFFFFFFF 00000320 FFFFFFFF FFFFFFFF

002F20 E3C5E7E3 40C4C5E2 C3D9C9D7 E3D6D940 TEXT DESCRIPTOR
002F30 04000000 00000080 00000006 0000000C ..... 0.....
002F40 000003A0 FFFFFFFF 00000280 030235A8 ... μ..... 0... y
002F50 00000042 00000001 00000005 00000042 ... â..... â
002F60 00000002 00000005 00000042 00000003 ... â..... â
002F70 00000005 00000042 00000004 00000005 ... â..... â
002F80 00000042 00000005 00000005 00000042 ... â..... â
002F90 00000006 00000005 6E6E6E6E 6E6E6E6E ... >>>>>>>>

```

Note that there are two Map Descriptors:

002FA0	D4C1D740	C4C5E2C3	D9C9D7E3	D6D94040	MAP DESCRIPTOR	← first MAP DESCRIPTOR
002FB0	05000000	00000040	00000001	00000008	.....	
002FC0	000003E0	000003E0	07D9C9D7	00000280	... \-... \- RIP... Ø	
002FD0	FFFFFFFF	00000001	00000001	6E6E6E6E	..... >>>>	
002FE0	D4C1D740	C4C5E2C3	D9C9D7E3	D6D94040	MAP DESCRIPTOR	← second MAP DESCRIPTOR
002FF0	05000000	00000070	00000006	00000008	..... Ø.....	
003000	00000450	FFFFFFFF	04D9C9D7	00000140	... &-... RIP...	
003010	00000280	00000800	00000001	00000900	... Ø.....	
003020	00000002	00000A00	00000003	00000B00	.....	
003030	00000004	00000C00	00000005	00000D00	.....	
003040	00000006	6E6E6E6E	6E6E6E6E	6E6E6E6E	... >>>>>>>>>>	
003050	D4C1D740	E3C1C2D3	C5404040	40404040	MAP TABLE	
003060	07000280	00000058	00000004	0000000C	... Ø... i.....	
003070	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFF0006	.....	← the end
003080	FFFFFFFF	000003E0	00010006	00000280	..... \-..... Ø	
003090	000003E0	00000006	00000140	FFFFFFFF	... \-.....	
0030A0	FFFF0006	FFFFFFFF	00000000	00000000	.....	
0030B0	00000000	00000000	00000000	00000000	.....	

## Descriptor Types

We have identified the following Descriptor Types:

00	View Space Header
01	Dummy Record
02	View Descriptor
03	File Descriptor
04	Text Descriptor
05	Map Descriptor
06	?
07	Map Table

## No Debug Source

If the module is compiled without any DBGVIEW as in:

```
CRTCMOD MODULE(I LEMAIN) SRCFILE(QCSRC) OUTPUT(*PRINT) OPTION(*NOSHOWINC)
```

the functional part of the associated space would *not* contain the address of the HLL symbol table:

Address	1104BEC0D6 000000			..... aØ · IR0»...	
000000	00020010 00818000	2976D9EE	8B000000		
000010	D0010000 00000000	1104BEC0	D6001000	}..... {0...	← back address to space
000020	00000000 FF000000	00000000	00000000	· ktl.....	← No HLL Symbol Table
000030	00000000 FF000000	00000000	00000000	.....	

## Why Make the Symbol Table Inaccessible?

<LAMENT>

It is a mystery why the HLL symbol table is stored in the functional part of the associated space where it cannot be accessed by an MI-instruction. How does OS/400 or the compilers access the symbol table? But then again: why place the \*MODULE in the System Domain in the first place? There seems to no real reason to do so.

</LAMENT>

## Changing Source/Debug Information, MI CHGMOD

We can now pull together what we have learned. The MI CHGMOD program will allow you to change the source library in the OIR *and* at the same time in the debug file descriptor (if any) for a \*MODULE. In this way the two sources of source/debug information are kept in sync. You can easily change other things as well, although the library is the most likely candidate (the “promotion” problem). You call the program giving the name of the module, of name the library where the \*MODULE resides, and the new library name:

```
===> CALL MI CHGMOD PARM(module library new-library)
```

First the parameters:

```
DCL SPCPTR .PARM1 PARM;
DCL DD PARM-MODULE          CHAR(10) BAS(.PARM1);

DCL SPCPTR .PARM2 PARM;
DCL DD PARM-OBJECT-LIBRARY CHAR(10) BAS(.PARM2);

DCL SPCPTR .PARM3 PARM;
DCL DD PARM-SOURCE-LIBRARY CHAR(10) BAS(.PARM3);

DCL OL PARMS (.PARM1, .PARM2, .PARM3) PARM EXT MIN(3);

ENTRY * (PARMS) EXT;
```

We need to use our valued **MI MAKPTR** helper-program to manufacture pointers from addresses. Also, the MI CHGMOD program must be a system-state program. We have here a perfect example of being forced to elevate a program way beyond what is reasonable; there should be no reason why we cannot implement MI CHGMOD using documented interfaces and user-state programs.

```
DCL SYSPTR .THE-CONTEXT;
DCL DD RESOLVE CHAR(34);
  DCL DD RESOLVE-TYPE CHAR( 2) DEF(RESOLVE) POS( 1);
  DCL DD RESOLVE-NAME CHAR(30) DEF(RESOLVE) POS( 3);
  DCL DD RESOLVE-AUTH CHAR( 2) DEF(RESOLVE) POS(33) INIT(X'0000');

RESOLVE-TO-HELPER-PGM:
  CPYBLA      RESOLVE-TYPE, X'0201';
  CPYBLAP     RESOLVE-NAME, "MI MAKPTR", " ";
  RSLVSP      .MI MAKPTR, RESOLVE, *, *;
```

We'll allow QTEMP to be the object-library:

```
DCL DD PC0 CHAR(256) BASPC0;
DCL SYSPTR .QTEMP DEF(PC0) POS(65);

DETERMINE-CONTEXT:
  CPYBWP      .THE-CONTEXT, .QTEMP;
  CMPBLAP(B)  PARM-OBJECT-LIBRARY, "QTEMP", " "/EQ(GET-MODULE);
  CPYBLA      RESOLVE-TYPE, X'0401';
  CPYBLAP     RESOLVE-NAME, PARM-OBJECT-LIBRARY, " ";
  RSLVSP      .THE-CONTEXT, RESOLVE, *, *;
```

We now locate the module in the context (*i.e.* library) that we have determined:

```
GET-MODULE:
  CPYBLA      RESOLVE-TYPE, X'0301';
  CPYBLAP     RESOLVE-NAME, PARM-MODULE, " ";
  RSLVSP      .SYSPTR, RESOLVE, .THE-CONTEXT, *;
```

The first order of business is to update the OIR. We could have called the program (MI WRKDIR) that we developed in Chapter 31, but as updating the OIR is only a few instructions, we do that here with a local subroutine:

```
FIRST-UPDATE-OIR:
  CALLI      CHANGE-OIR, *, .CHANGE-OIR;
```

Where (see the program for the data declarations needed to find and modify the OIR):

```
DCL INSPTR .CHANGE-OIR;
ENTRY CHANGE-OIR INT;
  SETSPFPF    .LIBSPC, .THE-CONTEXT;
  SETSPFPF    .SPC-OIR, .SYS-OIR;

RETRIEVE-OIR-DATA:
  CPYBLA      INDEX-SEARCH-ARG, X'0301'; /* MODULE */
  CPYBLA      INDEX-SEARCH-ARG(3:10), PARM-MODULE;
  FND1NXEN    .INDEX-DATA, .SYS-IDX, .INDEX-OPT, .INDEX-SEARCH-ARG;
  CMPNV(B)    INDEX-RTN-COUNT, 0/EQ(.CHANGE-OIR);

FOUND-OIR:
  MULT        OFFSET, INDEX-OFFSET, 512;
  SUBN(S)     OFFSET, 256;
  ADDSPP      .OIR, .SPC-OIR, OFFSET;

  CPYBLA      OIR-REF-SOURCE-LIBRARY, PARM-SOURCE-LIBRARY;
  B           .CHANGE-OIR;
```

Now get a pointer to the functional part (offset x'000000') of the associated space:

```
DCL SPCPTR .MAKPTR INIT(MAKPTR);
DCL DD MAKPTR CHAR(16) BDRY(16);
DCL SPCPTR .SPCPTR DEF(MAKPTR) POS(1);
DCL SYSPTR .SYSPTR DEF(MAKPTR) POS(1);
DCL DD PTR-ADDRESS CHAR(8) DEF(MAKPTR) POS(9);

DCL OL MI MAKPTR (.MAKPTR) ARG;
DCL SYSPTR .MI MAKPTR;

GET-ASSOCIATED-SPACE:
SETSPFP .SPCPTR, .SYSPTR;

GET-FUNCTIONAL-SPACE:
CPYBREP PTR-ADDRESS(6:3), X'00'; /* offset 0 => functional part */
CALLX .MI MAKPTR, MI MAKPTR, *;
```

The .SPCPTR space pointer is manufactured by MI MAKPTR. The pointer points to unspecified storage:

```
DCL DD STORAGE CHAR(16384) BAS(.SPCPTR);
```

We can now pick up the address of the HLL Symbol Table. If the address starts with a word of binary zeroes, no symbol table is present:

```
POINT-TO-HLL-SYMBOL-TABLE:
CPYBLA PTR-ADDRESS, STORAGE(33:8); /* 0020 */
CMPBLA(B) PTR-ADDRESS, X'00000000'/EQ(NOT-FOUND);
CALLX .MI MAKPTR, MI MAKPTR, *;
```

We are not interested in the symbol table *per se*, but in the “XPF” piece that follows:

```
DCL SPCPTR .XPF;
DCL DD OFFSET BIN(4);

POINT-TO-XPF-PIECE:
CPYBRA OFFSET, STORAGE(57:4); /* 0038 */
ADDN(S) OFFSET, 32;
ADDSP .XPF, .SPCPTR, OFFSET;
```

We created a special pointer to the XPF, because all descriptor offsets are calculated from the XPF base address. The View Space Header starts 32 bytes into the XPF, so:

```
POINT-TO-VIEW-SPACE-HDR:
ADDSP .SPCPTR, .XPF, 32;
```

And we can finally pick up the offset to the first descriptor and construct a space pointer to it:

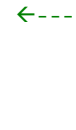
```
POINT-TO-FIRST-DESCRIPTOR:
CPYBRA OFFSET, STORAGE(65:4);
ADDSP .SPCPTR, .XPF, OFFSET;

DCL DD DESCRIPTOR CHAR(4096) BAS(.SPCPTR);
DCL DD DESCR-EYE-CATCHER CHAR(16) DEF(DESCRIPTOR) POS(1);
DCL DD DESCR-TYPE CHAR(1) DEF(DESCRIPTOR) POS(17);
DCL DD * CHAR(3) DEF(DESCRIPTOR) POS(18);
DCL DD DESCR-SIZE BIN(4) DEF(DESCRIPTOR) POS(21);
DCL DD * BIN(4) DEF(DESCRIPTOR) POS(25);
DCL DD * BIN(4) DEF(DESCRIPTOR) POS(29);
DCL DD DESCR-OFFSET BIN(4) DEF(DESCRIPTOR) POS(33);
DCL DD DESCR-LAST BIN(4) DEF(DESCRIPTOR) POS(37);
```

The plan is now to cycle through descriptors until a descriptor is found with a type of “File Descriptor”. The “chain” of descriptors ends when the offset to the next one is -1, so:

```
DCL DD END-OF-CHAIN BIN(4) INIT(-1);

CHECK-FOR-FILE-DESCRIPTOR:
CMPBLA(B) DESCR-TYPE, X'03'/EQ(FOUND-FILE-DESCRIPTOR);
NEXT-DESCRIPTOR:
CMPNV(B) DESCR-OFFSET, END-OF-CHAIN/EQ(NOT-FOUND);
ADDSP .SPCPTR, .XPF, DESCR-OFFSET;
B CHECK-FOR-FILE-DESCRIPTOR;
```



Instead of testing the descriptor type, we could also (maybe even in addition to testing the type) check if the “eye catcher” is “FILE DESCRIPTOR”. If we found a File Descriptor, but it is not the last one, we press on looking for the next:

```
FOUND-FILE-DESCRIPTOR:
  CMPNV(B)      DESCR-LAST,  END-OF-CHAIN/NEQ(NEXT-DESCRIPTOR);
```

Otherwise, we change the information as desired:

```
DCL DD FILE-DESCRIPTOR CHAR(4096) BAS(. SPCPTR);
DCL DD FILE-EYE-CATCHER  CHAR(16) DEF(FILE-DESCRIPTOR) POS( 1);
DCL DD *                  CHAR(32) DEF(FILE-DESCRIPTOR) POS(17);
DCL DD FILE-SOURCE-FILE  CHAR(10) DEF(FILE-DESCRIPTOR) POS(49);
DCL DD FILE-SOURCE-LIBRARY CHAR(10) DEF(FILE-DESCRIPTOR) POS(59);
DCL DD FILE-SOURCE-MEMBER CHAR(10) DEF(FILE-DESCRIPTOR) POS(69);

CPYBLA      FILE-SOURCE-LIBRARY,  PARM-SOURCE-LIBRARY;
B           RETURN;
```

If we do not find a descriptor, we ignore the change request (but we could have some other reaction):

```
NOT-FOUND:
```

```
RETURN:
  RTX      *;
```

## The Bound Program

When the modules are *bound* into a program (or a service program), the \*MODULE objects become *embedded* in the \*PGM (or \*SRVPGM) object and are no longer needed (nor individually available if the program is saved and then restored into a different environment or system). As we mentioned, the OIR is not be used to store the module source information, which end up *blank*:

Work with OIR			
Object name	ILEMAIN	Name	
Library	LSVALGAARD	Name	
Object type/subtype	0201 ← *pgm	Hexadecimal values	
Version/Release	V4R4M0	VxRxMy	
Source Library		Library name	
Source File		File name	
Source Member		Member name	

If we look at the program object:

```
Address 3170F0B7D1 000000
000000 00010020 00900001 3170F0B7 D1000000 ..... °... ø0%J...
000010 70010300 00000000 281A9F2E B2001000 ø..... ¢. ¥... ← associated space
000020 80000201 C9D3C5D4 C1C9D540 40404040 ø... ILEMAIN
000030 40404040 40404040 40404040 40404040
000040 40408000 00001000 000000E0 3F103600 ø..... \...
000050 82CBD3C2 32468000 053B6FC9 F9000000 bôLB- āø... ?I 9... ← OWNER
000060 00000000 00000000 01AA965C 51000000 ..... | o*é... ← CONTEXT
000070 3170F0B7 D1001000 10000000 01000000 . ø0%J.....
000080 82CBD3C7 04B40000 00000000 00000000 bôLG- ©.....
000090 00000000 00000000 00000000 00000000 .....
```

## The BNAS Information

We find that the associated space contains the module source information (probably garnered from the individual module OIR entries). The data seems to have a signature (“BNAS”) identifying it as (possibly) BouNd Associate Space information):

```
Address 281A9F2EB2 001000
001000 02039301 C2D5C1E2 00000003 00000020 .. I- BNAS .....
001010 00000000 00000000 00000000 00000000 .....
```

Several types of records are indicated:

```
001020 01000000 00010001 00000014 00000030 ..... ← type, size
001030 00050000 00000000 00000000 00000000 .....
```

001040	00000000	00000000	00000000	00000000	.....
001050	01000000	00010002	00000370	00000380	..... 0... 0 ← type, size
001060	00000370	0003D8C2	D5D9E3E5	D6C9E2E5	... 0- QBNRTVOI SV
001070	00014040	40404040	40404040	40404040	..
001080	40404040	40404040	40404040	40404040	
001090	00000000	00000000	00000000	00000000	.....

This record contains the original OIR information for the modules:

0010A0	C9D3C5D4	C1C9D540	4040D3E2	E5C1D3C7	I LEMAIN LSVALG
0010B0	C1C1D9C4	E0C3D3C5	40404040	40404040	AARD\CLE
0010C0	40404040	40404040	40000000	00000000	.....
0010D0	00000000	005CC8C5	E7404040	40404040	..... *HEX
0010E0	40404040	40404040	405CD1D6	C2D9E4D5	*JOB RUN
0010F0	40404000	00000000	00000001	01800000	..... 0..
001100	00000000	00000000	40404040	40404040	.....
001110	40404040	40404040	40404040	40F14040	1
001120	40404040	4040F5F7	F6F9C3E7	F2E5F4D9	5769CX2V4R
001130	F4D4F0E5	F4D9F4D4	F0F1F0F1	F0F7F2F3	4MOV4R4M01010723
001140	F0F8F1F4	F2F8F1F0	F1F0F6F2	F3F1F0F2	0814281010623102
001150	F0F2F6D8	C3E2D9C3	40404040	40C9D3C5	026QCSRC I LE
001160	D4C1C9D5	404040D3	E2E5C1D3	C7C1C1D9	MAIN LSVALGAAR
001170	C4404040	40404040	40404040	40404040	D
001180	00000000	00000000	00000000	00000000	.....
001190	00000000	00000000	0000F1F6	F0000000	..... 160..
0011A0	40400000	00000000	00000000	00000000	.....
0011B0	C9D3C5D4	D6C4F140	4040D3E2	E5C1D3C7	I LEMOD1 LSVALG
0011C0	C1C1D9C4	E0C3D3C5	40404040	40404040	AARD\CLE
0011D0	40404040	40404040	40000000	00000000	.....
0011E0	00000000	005CC8C5	E7404040	40404040	..... *HEX
0011F0	40404040	40404040	405CD1D6	C2D9E4D5	*JOB RUN
001200	40404000	00000000	00000002	01000000	.....
001210	00000000	00000000	40404040	40404040	.....
001220	40404040	40404040	40404040	40F14040	1
001230	40404040	4040F5F7	F6F9C3E7	F2E5F4D9	5769CX2V4R
001240	F4D4F0E5	F4D9F4D4	F0F1F0F1	F0F7F2F3	4MOV4R4M01010723
001250	F0F8F1F4	F2F1F1F0	F1F0F6F2	F3F1F0F2	0814211010623102
001260	F0F4F9D8	C3E2D9C3	40404040	40C9D3C5	049QCSRC I LE
001270	D4D6C4F1	404040D3	E2E5C1D3	C7C1C1D9	MOD1 LSVALGAAR
001280	C4404040	40404040	40404040	40404040	D
001290	00000000	00000000	00000000	00000000	.....
0012A0	00000000	00000000	00000000	00000000	.....
0012B0	40400000	00000000	00000000	00000000	.....
0012C0	C9D3C5D4	D6C4F240	4040D3E2	E5C1D3C7	I LEMOD2 LSVALG
0012D0	C1C1D9C4	E0C3D3C5	40404040	40404040	AARD\CLE
0012E0	40404040	40404040	40000000	00000000	.....
0012F0	00000000	005CC8C5	E7404040	40404040	..... *HEX
001300	40404040	40404040	405CD1D6	C2D9E4D5	*JOB RUN
001310	40404000	00000000	00000003	01000000	.....
001320	00000000	00000000	40404040	40404040	.....
001330	40404040	40404040	40404040	40F14040	1
001340	40404040	4040F5F7	F6F9C3E7	F2E5F4D9	5769CX2V4R
001350	F4D4F0E5	F4D9F4D4	F0F1F0F1	F0F7F2F3	4MOV4R4M01010723
001360	F0F8F1F4	F2F4F1F0	F1F0F6F2	F3F1F0F2	0814241010623102
001370	F1F1F4D8	C3E2D9C3	40404040	40C9D3C5	114QCSRC I LE
001380	D4D6C4F2	404040D3	E2E5C1D3	C7C1C1D9	MOD2 LSVALGAAR
001390	C4404040	40404040	40404040	40404040	D
0013A0	00000000	00000000	00000000	00000000	.....
0013B0	00000000	00000000	00000000	00000000	.....
0013C0	40400000	00000000	00000000	00000000	.....

Finally there is a list of service programs used:

0013D0	01000000	00010003	000000A0	00000000	..... μ... ← type, no more
0013E0	000000A0	00000003	00010000	00000000	... μ..... ← number of srvgms
0013F0	D8C3F2C9	D6404040	4040D8E2	E8E24040	QC2I0 QSYS <== SERVICE PGM 1
001400	40404040	00000005	00030000	00000000	.....
001410	6E2D63C4	2B8D16E7	4EA1DD08	E3AF52EF	>. ÄD. ý. X+~ü. T®ëÖ <== SIGNATURE FOR 1
001420	D8C3F2E4	E3C9D3F1	4040D8E2	E8E24040	QC2UTI L1 QSYS <== SERVICE PGM 2
001430	40404040	00000006	00030000	00000000	.....
001440	CB1F8035	82C31AC2	5B4DCBDD	1B6D0948	ô. Ø. bC. B\$(ôü. _ . ç <== SIGNATURE FOR 2
001450	D8D3C5C1	E6C94040	4040D8E2	E8E24040	QLEAWI QSYS <== SERVICE PGM 3
001460	40404040	0000000D	00030000	00000000	.....
001470	44F70FAB	A0858539	7BDF0CF1	95F82EC1	à7. çµee. #ý. 1n8. A <== SIGNATURE FOR 3
001480	00000000	00000000	00000000	00000000	.....



Address	<b>281A9F2EB2</b>	<b>000000</b>			
000000	00020010	00810001	3170F0B7	D1000000	. . . . . a . . . ø0¼J . .
000010	50010000	00000000	281A9F2E	B2001000	& . . . . . ¢ . ¥ . .
000020	<b>0FA88400</b>	<b>FD001000</b>	00000000	000001A0	. yd. Û . . . . . µ ← V2R3 debug rel ic mod1
000030	<b>0FA88400</b>	<b>FD0011A0</b>	00000000	000001A0	. yd. Û . µ . . . . . µ ← V2R3 debug rel ic mod2
000040	<b>0FA88400</b>	<b>FD001340</b>	00000000	000001A0	. yd. Û . . . . . µ ← V2R3 debug rel ic mod3
000050	<b>373668D3</b>	<b>48001000</b>	00000000	00000000	. ÇLÇ . . . . . ← HLL STBL 1 <sup>st</sup> module
000060	<b>1EC4DA5</b>	<b>93001000</b>	00000000	00000000	. D(vI . . . . . ← HLL STBL 2 <sup>nd</sup> modul e
000070	<b>06523277</b>	<b>DE001000</b>	00000000	00000000	. ê. Ŧú . . . . . ← HLL STBL 3 <sup>rd</sup> modul e
000080	00000000	00000000	00000000	00000000	. . . . .
000090	00000000	00000000	00000000	00000000	. . . . .

and similarly for the other two modules

38-10



and similarly for the other two modules.

## Debug Source Information

38-11

## Appendix A

### MI-Instructions Quick Reference

#### *MI-Instruction Quick Reference*

The following Quick Reference Guide covers all MI-instructions for which information for the AS/400 is available. It is not meant as a substitute for the MI Functional Reference, but does provide a useful (and mercifully short) overview of the various instructions, their possible operands and branch conditions. I use it all the time in lieu of the MIFR. I hope the format is reasonably self-explanatory.

#### Computational and Branching Instructions

OPCODE	Name	Operands and Conditions
ADDLC IBS  4 ext	Add Logical Character	Sum CHARVF Addend 1 CHARF Addend 2 CHARF All same length <= 256 ZNTC NTZNTC ZC NTZC
ADDN IBSR  4 ext	Add Numeric	Sum NUMV Addend 1 NUM Addend 2 NUM POS NEG ZER NAN
AND IBS  2 ext	And Bytes	Result CHARV Source 1 CHAR Source 2 CHAR Longer of 2 Sources, then place in result, pad X'00' or truncate ZER NZER
B	Branch	Target BP, INSPTR, IDL element
CIPHER	Cipher	Result SPCPTR Control CHAR(32)V Source SPCPTR
CIPHERKY	Cipher Key	Result CHAR(8)V Control CHAR(64)V Source CHAR(8)
CLRBTS	Clear Bit in String	Result CHARV, NUMV Bit number BIN
CMPBLA IB req  3 ext	Compare Bytes Left- Adjusted	Compare 1 CHAR, NUM Compare 2 CHAR, NUM Shorter of 2 compare operands Stops on 1st unequal byte HI LO EQ
CMPBLAP IB req  3 ext	Compare Bytes Left- Adjusted with Pad	Compare 1 CHAR, NUM Compare 2 CHAR, NUM Pad CHAR(1), NUM(1) Shorter of 2 compare operands Stops on 1st unequal byte HI LO EQ
CMPBRA IB req  3 ext	Compare Bytes Right-Adjusted	Compare 1 CHAR, NUM Compare 2 CHAR, NUM Shorter of 2 compare operands Stops on 1st unequal byte HI LO EQ

OPCODE	Name	Operands and Conditions
CMPBRAP IB req  3 ext	Compare Bytes Right-Adjusted with Pad	Compare 1 CHAR, NUM Compare 2 CHAR, NUM Pad CHAR(1), NUM(1) Shorter of 2 compare operands Stops on 1st unequal byte HI LO EQ
CMPNV IB req 4 ext	Compare Numeric Values	Compare 1 NUM Compare 2 NUM HI LO EQ UNOR
CMPSW IB req  1 ext	Compare and Swap	Compare 1 CHAR(1, 2, 4, 8) V Compare 2 CHAR(1, 2, 4, 8) V Swap Operand CHAR(1, 2, 4, 8) EQ
CPRDATA	Compress Data	Template SPCPTR to CHAR(64)
CAI	Compute Array Index	Index BIN(2) V Subscript A BIN(2) Subscript B BIN(2) Dimension BIN(2) C, IMM(2)
CMF1 IB  4 ext	Compute Mathematical Function using 1 Input	Result FLTV Control CHAR(2) Source FLT POS NEG ZER NAN
CMF2 IB  4 ext	Compute Mathematical Function using 2 Inputs	Result FLTV Control CHAR(2) Source 1 FLT Source 2 FLT POS NEG ZER NAN
CAT	Concatenate	Result CHARV Source 1 CHAR Source 2 CHAR Length of Result, pad X'40' or truncate
CVTBC IB  3 ext	Convert BSC to Characters	Result CHARV Control CHAR(3) VF Source CHAR CR SE TR
CVTCB IB  2 ext	Convert Characters to BSC	Result CHARV Control CHAR(3) VF Source CHAR SE RO
CVTCH	Convert Characters to HEX	Result CHARV Source CHAR [0-9A-F] Length of Result, pad X'00'
CVTCM IB  2 ext	Convert Characters to Multi-leaving Remote Job Entry Format	Result CHARV Control CHAR(13) VF Source CHAR SE RO
CVTCN	Convert Characters to Numeric	Result NUMV, DTAPTR-NUM Source CHAR, DTAPTR-CHAR Attributes CHAR(7), DTAPTR-CHAR(?)
CVTCS IB  2 ext	Convert Characters to SNA Format	Result CHARV Control CHAR(15) V Source CHAR SE RO
CVTDFFP	Convert Decimal Form to Floating-Point	Result FLOATV Dec.exponent PKD, ZND Dec.mantissa PKD, ZND

OPCODE	Name	Operands and Conditions
CVTEFN	Convert External Form to Numeric Value	Result NUMV, DTAPTR-NUM Source CHAR, DTAPTR-CHAR Mask CHAR(3), DTAPTR-CHAR(3), *
CVTFPDF R	Convert Floating-Point to Decimal Form	Dec.exponent PKDV, ZNDV Dec.mantissa PKDV, ZNDV Source FLOAT
CVTHC	Convert HEX to Characters	Result CHARV Source CHAR Length of Result, pad X'F0'
CVTMC IB 2 ext	Convert Multi-leaving Remote Job Entry Format to Characters	Result CHARV Control CHAR(6)VF Source CHAR SE RO
CVTNC	Convert Numeric to Characters	Result CHARV, DTAPTR-CHARV Source NUM, DTAPTR-NUM Attributes CHAR(7), DTAPTR-CHAR(7)
CVTSC IB 3 ext	Convert SNA Format to Characters	Result CHARV Control CHAR(14)V Source CHAR SE RO EC
CPYBTA	Copy Bits Arithmetic	Receiver CHARV, NUMV Source CHAR, NUM Offset IMM Length IMM between 1 and 32 Length of Receiver not > than 4 bytes
CPYBTL	Copy Bits Logical	Receiver CHARV, NUMV Source CHAR, NUM Offset IMM Length IMM between 1 and 32 Length of Receiver not > than 4 bytes
CPYBTLLS	Copy Bits with Left Logical Shift	Receiver CHARV, NUMV Source CHAR, NUM, IMM(1) Shift CHAR(2)F, IMM(2) Length of Receiver, pad B'0'
CPYBTRAS	Copy Bits with Right Arithmetic	Receiver CHARV, NUMV Source CHAR, NUM, IMM(1) Shift CHAR(2)F, IMM(2) Length of Receiver, pad sign
CPYBTRLS	Copy Bits with Right Logical Shift	Receiver CHARV, NUMV Source CHAR, NUM, IMM(1) Shift CHAR(2)F, IMM(2) Length of Receiver, pad B'0'
CPYBLA	Copy Bytes Left-Adjusted	Receiver CHARV, NUMV, DTAPTR-CHARV, -NUMV Source CHAR, NUM, DTAPTR-CHAR, -NUM Shorter of two operands
CPYBLAP	Copy Bytes Left-Adjusted with Pad	Receiver CHARV, NUMV, DTAPTR-CHARV, -NUMV Source CHAR, NUM, DTAPTR-CHAR, -NUM Pad CHAR(1), NUM(1) Length of Receiver, pad
CPYBOLA	Copy Bytes with Overlap Left-Adjusted	Receiver CHARV, NUMV, Source CHAR, NUM, Shorter of two operands

OPCODE	Name	Operands and Conditions
CPYBOLAP	Copy Bytes with Overlap Left-Adjusted with Pad	Receiver CHARV, NUMV, Source CHAR, NUM, Pad CHAR(1), NUM(1) Length of Receiver, pad
CPYBREP	Copy Bytes Repeatedly	Receiver CHARVF, NUMV, Source CHARF, NUM, Length of Receiver
CPYBRA	Copy Bytes Right- Adjusted	Receiver CHARV, NUMV, DTAPTR-CHARV, -NUMV Source CHAR, NUM, DTAPTR-CHAR, -NUM Shorter of two operands
CPYBRAP	Copy Bytes Right- Adjusted with Pad	Receiver CHARV, NUMV, DTAPTR-CHARV, -NUMV Source CHAR, NUM, DTAPTR-CHAR, -NUM Pad CHAR(1), NUM(1) Length of Receiver, pad
CPYBBTA	Copy Bytes to Bits Arithmetic	Receiver CHARV, NUMV, Offset IMM Length IMM from 1 to 32 Source CHAR, NUM, Length of Source <= 4 bytes
CPYBBTL	Copy Bytes to Bits Logical	Receiver CHARV, NUMV, Offset IMM Length IMM from 1 to 32 Source CHAR, NUM, Length of Source <= 4 bytes
CPYECLAP	Copy Extended Characters Left-Adjusted with Pad	Receiver DTAPTR-CHARV Source DTAPTR-CHAR Pad CHAR(3), *
CPYHEXNN	Copy Hex Digit Numeric to Numeric	Receiver CHARV, NUMV, Source CHARF, NUM,
CPYHEXNZ	Copy Hex Digit Numeric to Zone	Receiver CHARVF, NUMV Source CHARF, NUM
CPYHEXZN	Copy Hex Digit Zone to Numeric	Receiver CHARVF, NUMV Source CHARF, NUM
CPYHEXZZ	Copy Hex Digit Zone to Zone	Receiver CHARVF, NUMV Source CHARF, NUM
CPYNV IBR 4 ext	Copy Numeric Value	Receiver NUMV, DTAPTR-NUMV Source NUM, DTAPTR-NUM POS NEG ZER NAN
DCPDATA	Decompress Data	Template SPCPTR to CHAR(64)
DIV IBSR 4 ext	Divide	Quotient NUMV Dividend NUM Divisor NUM POS NEG ZER NAN
DIVREM IBSR 3 ext	Divide with Remainder	Quotient NUMV Dividend NUM Divisor NUM Remainder NUMV POS NEG ZER
EDIT	Edit	Result CHARV, DTAPTR-CHARV Source NUM, DTAPTR-NUM Mask NUM, DTAPTR-NUM
EXCHBY	Exchange Bytes	Source 1 NUMV, CHARVF Source 2 NUMV, CHARVF Same length

OPCODE	Name	Operands and Conditions
XOR IBS  2 ext	Exclusive Or Bytes	Result CHARV Source 1 CHAR Source 2 CHAR Longer of 2 Sources, then place in result, pad X'00' or truncate ZER NZER
ECSCAN IB req  3 ext	Extended Character Scan	Result BINV, BINA Base CHAR Compare CHAR Mode CHAR(1) POS ZER EC
EXTREXP IB 4 ext	Extract Exponent	Result BINV Source FLOAT NOR DEN INF NAN
EXTRMAG IB 3 ext	Extract Magnitude	Result NUMV Source NUM POS ZER NAN
MULT IBSR  4 ext	Multiply	Product NUMV Multiplicand NUM Multiplier NUM POS NEG ZER NAN
NEG IBS 4 ext	Negate Number	Result NUMV Source NUM POS NEG ZER NAN
NOT IBS  2 ext	Not Bytes	Result CHARV Source CHAR Length of Result, pad X'00' ZER NZER
OR IBS  2 ext	Or Bytes	Result CHARV Source 1 CHAR Source 2 CHAR Longer of 2 Sources, then place in result, pad X'00' or truncate ZER NZER
REM IBS  3 ext	Remainder	Remainder NUMV Dividend NUM Divisor NUM POS NEG ZER
SCALE IBS  4 ext	Scale	Result NUMV Source NUM Scale Factor BIN(2) POS NEG ZER NAN
SCAN IB  3 ext	Scan Characters	Result BINV, BINA Base CHAR (<= 32k) Compare CHAR POS ZER NCMP
SCANWC IB  4 ext	Scan with Control	Base Locator SPCPTR Control CHAR(8)V Options CHAR(4) Escape Addr BP, INSPTR, IDL elem, * HI LO EQ NF
SEARCH IB  2 ext	Search	Result BINV, BINA Array CHARA, NUMA Find CHAR, NUM Location BIN POS ZER
SETBTS	Set Bit in String	Result CHARV, NUMV Offset BIN

OPCODE	Name	Operands and Conditions	
SETIP	Set Instruction Pointer	Result	INSPTR
		Target	BP
SSCA IB	Store and Set Computational Attributes	Old Attrs	CHAR(5)V
		New Attrs	CHAR(5), *
		Control	CHAR(5), *
SUBLC IBS  3 ext	Subtract Logical Character	Difference	CHARVF
		Minuend	CHARF
		Subtrahend	CHARF
		All same length <= 256	
		ZC NTZC NTZNTC	
SUBN IBSR  4 ext	Subtract Numeric	Difference	NUMV
		Minuend	NUM
		Subtrahend	NUM
		POS NEG ZER NAN	
TSTRPLC IBS	Test and Replace Characters	Result	CHARV
		Replacement	CHAR
TSTBTS IB 2 ext	Test Bit in String	Source	CHARV, NUMV
		Offset	BIN
		ZER ONE	
TSTBUM IB req  3 ext	Test Bits Under Mask	Source	CHAR, NUM
		Mask	CHAR, NUM
		Only 1st byte	
		ZER ONES MXD	
XLATE	Translate	Result	CHARV
		Source	CHAR
		Position	CHAR, *
		Replacement	CHAR
XLATEMB	Translate Multiple Bytes	Template	SPCPTR to CHAR(128)
XLATEWT	Translate with Table	Result	CHARV
		Source	CHAR
		Table	CHAR(256)
		Shorter of Result and Source	
XLATWTDS	Translate with Table and DBCS Skip	Target	CHARV
		Length	BIN(4)
		Table	CHAR(256)
TRIML	Trim Length	Length	NUMV
		Source	CHAR
		Trim Char	CHAR(1)
VERIFY IB  2 ext	Verify	Invalid pos.	BINV, BINA
		Source	CHAR
		Class	CHAR
		ZER POS	

## Pointer/Resolution Instructions

OPCODE	Name	Operands and Conditions
CMPPTRA IB req 2 ext	Compare Pointers for Object Addressability	Compare 1 PTR Compare 2 PTR EQ NEQ
CMPSPAD IB req  4 ext	Compare Pointers for Space Addressability	Compare 1 SPCPTR, DTAPTR Compare 2 SPCPTR, DTAPTR, NUMV, NUMA, CHARV, CHARA HI LO EQ UNEQ
CMPPTRE IB req 2 ext	Compare Pointers for Equality	Compare 1 PTR Compare 2 PTR EQ NEQ
CMPPTRT IB req 2 ext	Compare Pointer Types	Compare PTR Type CHAR(1), * EQ NEQ
CPYBWP	Copy Bytes with Pointer	Receiver CHARV, PTR Source CHAR, PTR, * Shorter of Receiver and Source
MATPTR	Materialize Pointer	Receiver SPCPTR Pointer PTR
MATPTRIF	Materialize Pointer Information	Receiver SPCPTR Pointer PTR Mask CHAR(4)
MATPTRL	Materialize Pointer Locations	Receiver SPCPTR Source SPCPTR Length BIN
MATCTX	Materialize Context	Receiver SPCPTR Context SYSPTR, * Options CHARF
RSLVDP	Resolve Data Pointer	Pointer DTAPTR Object CHAR(32)F, * Program SYSPTR, *
RSLVSP	Resolve System Pointer	Pointer SYSPTR Object CHAR(34)F, * Context SYSPTR, * Authority CHAR(2)F, *
ADDSP	Add Space Pointer	Result SPCPTR Source SPCPTR Increment BIN
CMPSPAD IB req  4 ext	Compare Space Addressability	Compare 1 NUMV, NUMA, CHARV, CHARA, PTR, PTR Compare 2 NUMV, NUMA, CHARV, CHARA, PTRDOA HI LO EQ UNEQ
SETDP	Set Data Pointer	Result DTAPTR Source NUMV, NUMA, CHARV, CHARA
SETDPADR	Set Data Pointer Addressability	Result DTAPTR Source NUMV, NUMA, CHARV, CHARA
SETDPAT	Set Data Pointer Attributes	Result DTAPTR Attributes CHAR(7)F
SETSP	Set Space Pointer	Result SPCPTR Source NUMV, NUMA, CHARV, CHARA, PTRDO
SETSPPD	Set Space Pointer with Displacement	Result SPCPTR Source NUMV, NUMA, CHARV, CHARA, PTRDO Displacement BIN



OPCODE	Name	Operands and Conditions	
SETSPFP	Set Space Pointer from Pointer	Result	SPCPTR
		Source	DTAPTR, SYSPTR, SPCPTR
SETSPPO	Set Space Pointer Offset	Result	SPCPTR
		Source	BIN
SETSPFP	Set System Pointer from Pointer	Result	SYSPTR
		Source	DTAPTR, SYSPTR, SPCPTR, INSPTR
STSPPO	Store Space Pointer Offset	Result	BINV
		Source	SPCPTR
SUBSPP	Subtract Space Pointer Offset	Result	SPCPTR
		Source	SPCPTR
		Decrement	BIN
SUBSPFO	Subtract Space Pointers For Offset	Offset Diff	BIN(4)V
		Pointer	SPCPTR
		Pointer	SPCPTR

## Space Management Instructions

OPCODE	Name	Operands and Conditions	
CRTS	Create Space	Space	SYSPTR
		Template	SPCPTR
DESS	Destroy Space	Space	SYSPTR
MATS	Materialize Space Attributes	Result	SPCPTR
		Space	SYSPTR
MODS	Modify Space Attributes	Space	SYSPTR
		Template	BIN, CHAR(28)

## Independent Index Instructions

OPCODE	Name	Operands and Conditions	
CRTINX	Create Independent Index	Index	SYSPTR
		Template	SPCPTR
DESINX	Destroy Independent Index	Index	SYSPTR
FNDINXEN	Find Independent Index Entry	Result	SPCPTR
		Index	SYSPTR
		Options	SPCPTR
		Argument	SPCPTR
INSINXEN	Insert Independent Index Entry	Index	SYSPTR
		Argument	SPCPTR
		Options	SPCPTR
RMVINXEN	Remove Independent Index Entry	Receiver	SPCPTR, *
		Index	SYSPTR
		Options	SPCPTR
		Argument	SPCPTR
MATINXAT	Materialize Independent Index Attributes	Receiver	SPCPTR
		Index	SYSPTR
MODINX	Modify Independent Index	Index	SYSPTR
		Modifs	CHAR(4)

## Authorization Instructions

OPCODE	Name	Operands and Conditions	
MATAU	Materialize Authority	Result	SPCPTR
		Object	SYSPTR
		User Profile	SYSPTR, *
MATAUOBJ	Materialize Authorized Objects	Result	SPCPTR
		Object	SYSPTR
		Options	CHAR(1)F, CHAR(*)

OPCODE	Name	Operands and Conditions	
MATAL	Materialize Authority List	Receiver List Options	SPCPTR SYSPTR SPCPTR
MATAUU	Materialize Authorized Users	Result Object Options	SPCPTR SYSPTR CHAR(1)F
MATUP	Materialize User Profile	Result User Profile	SPCPTR SYSPTR
MATUPID	Materialize User Profile Pointers from ID	Result Template	SPCPTR SPCPTR
MODINVAU	Modify Inv. Auth. Attrs	Template	CHAR(1)
TESTAU 1B  2 ext	Test Authority	Result Object User Profile AUTH	CHAR(2)V, * SYSPTR, PTRDO CHAR(2)F NAUTH
TESTEAU 1B  2 ext	Test Authority Extended	Receiver Authority Invocation AUTH	CHAR(8)V, * CHAR(8) BIN(2), * NAUTH
TESTULA 1B  2 ext	Test User List Authority	Receiver System Obj. Authority AUTH	SPCPTR, * SYSPTR SPCPTR NAUTH

## Program and Invocation Instructions

OPCODE	Name	Operands and Conditions	
MATBPGM	Materialize Bound Program	Result Program	SPCPTR SYSPTR
MATPG	Materialize Program	Result Program	SPCPTR SYSPTR
ACTBPGM	Activate Bound Program	Template Program	SPCPTR SYSPTR
ACTPG	Activate Program	Program Program	SPCPTR, SYSPTR, PTRDO SYSPTR
CALLX	Call External	Program Arguments Returns	SYSPTR, PTRDO OL, * IDL, *
CALLI	Call Internal	Entry Point Arguments Return	ENTRY OL, * INSPTR
CLRIEXIT	Clear Invocation Exit		
DEACTPG	De-Activate Program	Program	SYSPTR, *
PEND	Program End		
FNDRINVN	Find Relative Invocation Number	Inv Number Range Criterion	BIN(4)V CHAR(48)F, * SPCPTR
MATACTAT	Materialize Activation Attributes	Receiver Act. Mark Selection	SPCPTR UBIN(4) CHAR(1)
MATAGPAT	Materialize Activation Group Attributes	Receiver Act. Mark Selection	SPCPTR UBIN(4) CHAR(1)
MATINV	Materialize Invocation	Receiver Selection	SPCPTR SPCPTR
MATINVAT	Materialize Invocation Attributes	Receiver Invocation Selection	SPCPTR CHAR(48)F, * SPCPTR
MATINVE	Materialize Invocation Entry	Receiver	SPCPTR

OPCODE	Name	Operands and Conditions
		Selection CHAR(8)F, * Options CHAR(1)F, *
MATINVS	Materialize Invocation Stack	Receiver SPCPTR Process SYSPTR, *
MODASA	Modify Automatic Storage Allocation	Storage PTRDO, * Size BIN
NOOP	No Operation	
NOOPS	No Operation and Skip	Skip Count UIMM
OVRPGATR	Override Program Attributes	Attr ID UIMM Modifier UIMM
RINZSTAT	Reinitialize Static Storage	Activation SPCPTR to Template
RTX	Return from External	Return BIN(2), *
SETALLEN	Set Argument List Length	Arguments OL Length BIN
SETIEXIT	Set Invocation Exit	Program SYSPTR Arguments OL, *
STPLLEN	Store Parameter List	Length BINV
XCTL	Transfer Control	Program SYSPTR, SPCPTR Arguments OL, *
MATPRATR	Materialize Process Attributes	Result SPCPTR Process SYSPTR, * Options CHAR(1)
MATPRAGP	Materialize Process Activation Group	Receiver SPCPTR
WAITTIME	Wait on Time	Wait CHAR(16)
YIELD	Yield Timeslice	

## Exception Management Instructions

OPCODE	Name	Operands and Conditions
MATEXCPD	Materialize Exception Description	Result SPCPTR Description EXCM Options CHAR(1)
MODEXCPD	Modify Exception Description	Description EXCM Modifs SPCPTR, CHAR(2)C Options CHAR(1)
RETEXCPD	Retrieve Exception Data	Result SPCPTR Options CHAR(1)F
RTNEXCP	Return from Exception	Invocation SPCPTR
SNSEXCPD	Sense Exception Description	Receiver SPCPTR Invocation SPCPTR to Template Exception SPCPTR to Template
SIGEXCP IB 2 ext	Signal Exception	Signal SPCPTR Exception SPCPTR IGN DEF
TESTEXCP IB 1 ext	Test Exception	Receiver SPCPTR Description EXCM SIG

## Queue Management Instructions

OPCODE	Name	Operands and Conditions
CRTQ	Create Queue	Queue SYSPTR Template SPCPTR

OPCODE	Name	Operands and Conditions	
DEQ IB 2 ext	Dequeue	Prefix Message Queue DQ NDQ	CHAR SPCPTR SYSPTR
DESQ	Destroy Queue	Queue	SYSPTR
ENQ	Enqueue	Queue Prefix Message	SYSPTR CHAR SPCPTR
MATPRMSG	Materialize Process Message	Receiver Message Source Selection	SPCPTR SPCPTR SPCPTR SPCPTR
MATQAT	Materialize Queue Attributes	Result Queue	SPCPTR SYSPTR
MATQMSG	Materialize Queued Messages	Result Queue Selection	SPCPTR SYSPTR CHAR(16)

## Object Lock Instructions

OPCODE	Name	Operands and Conditions	
LOCK	Lock Objects	Objects	SPCPTR
LOCKOL	Lock Object Location	Template	SPCPTR
LOCKSL	Lock Space Location	Location	PTRDO
		Lock Type	CHAR(1), *
MATAOL	Materialize Allocated Object Locks	Receiver	SPCPTR
		Object	SYSPTR, PTRDO
MATDRECL	Materialize Data Space Record Locks	Receiver	SPCPTR
		Selection	SPCPTR
MATOBJLK	Materialize Object Locks	Receiver	SPCPTR
		Object	SYSPTR, PTRDO
MATPRLK	Materialize Process Locks	Receiver	SPCPTR
		Process	SYSPTR, *
MATPRECL	Materialize Process Record Locks	Receiver	SPCPTR
		Selection	SPCPTR
MATSELLK	Materialize Selected Locks	Receiver	SPCPTR
		Object	SYSPTR, PTRDO
XFRLOCK	Transfer Object Lock	Process	SYSPTR
		Template	SPCPTR
UNLOCKOL	Unlock Object Location	Template	SPCPTR
UNLOCKSL	Unlock Space Location	Space	PTRDO
		Lock Type	CHAR(1), *

## Context Management Instructions

OPCODE	Name	Operands and Conditions	
CRTMTX	Create Pointer-Based Mutex	Mutex	SPCPTR
		Template	SPCPTR
		Result	BIN(4)V
DESMTX	Destroy Pointer-Based Mutex	Mutex	SPCPTR
		Options	SPCPTR
		Result	BIN(4)V
LOCKMTX	Lock Pointer-Based Mutex	Mutex	SPCPTR
		Template	SPCPTR
		Result	BIN(4)V
UNLKMTX	Unlock Pointer-Based Mutex	Mutex	SPCPTR
		Result	BIN(4)V

## Heap Management Instructions

OPCODE	Name	Operands and Conditions	
ALCHSS	Allocate Heap Space Storage	Heap Space	SPCPTR
		Heap ID	BIN(4)V
		Size	BIN(4)
CRTHS	Create Heap Space	Heap ID	BIN(4)V
		Template	SPCPTR
DESHSS	Destroy Heap Space	Heap ID	BIN(4)V
FREHSS	Free Heap Space Storage	Heap Space	SPCPTR
FREHSSMK	Free Heap Space Storage from Mark	Mark ID	PTRDO
MATHSAT	Materialize Heap Space Attributes	Receiver	SPCPTR
		Heap ID	SPCPTR to Template
		Selection	CHAR(1)
REALCHSS	Reallocate Heap Space Storage	Heap Space	SPCPTR
		Size	BIN(4)
SETHSSMK	Set Heap Space Storage Mark	Mark ID	PTRDO
		Heap ID	BIN(4)

## Resource Management Instructions

OPCODE	Name	Operands and Conditions
CRMD	Compute Resource Management Data	Result SPCPTR Source SPCPTR Control CHAR(8)
ENSOBJ	Ensure Object	Object SYSPTR
MATAGAT	Materialize Access Group Attributes	Receiver SPCPTR Access Group SYSPTR
MATRMD	Materialize Resource Management Data	Receiver SPCPTR Control CHAR(8)
SETACST	Set Access State	Template SPCPTR
MATDMPS	Materialize Dump Space	Receiver SPCPTR Dump Space SYSPTR
MATJPAT	Materialize Journal Port Attributes	Receiver SPCPTR Journal Port SYSPTR, PTRDO
MATJPAT	Materialize Journal Space Attributes	Receiver SPCPTR Journal SPC SYSPTR
MATINAT	Materialize Instruction Attributes	Receiver SPCPTR Selection CHAR(16)
MATSOBJ	Materialize System Object	Receiver SPCPTR Object SYSPTR
MATMATR	Materialize Machine Attributes	Receiver SPCPTR Selection CHAR(2), SPCPTR
MATMDATA	Materialize Machine Data	Receiver CHARV Options CHAR(2), UBIN(2), UIMM
TESTTOBJ IB 2 ext	Test Temporary Object	Object SYSPTR  EQ NEQ

## MI Support Functions Instructions

OPCODE	Name	Operands and Conditions
GENUUID	Generate Universal Unique Identifier	Result SPCPTR
DIAG	Diagnose	Function BIN Argument SPCPTR

## Date/Time/Timestamp Instructions

OPCODE	Name	Operands and Conditions
CDD	Compute Date Duration	Duration PKDV Date to CHAR Date from CHAR Template SPCPTR
CTSD	Compute Time Duration	Duration PKDV Time to CHAR Time from CHAR Template SPCPTR
CTSD	Compute Timestamp Duration	Duration PKDV Timestamp to CHAR Timestamp fr CHAR Template SPCPTR
CVTD	Convert Date	Result date CHARV, PKDV, ZNDV Source date CHAR, PKD, BIN Template SPCPTR
CVTT	Convert Time	Result time CHARV Source time CHAR Template SPCPTR

OPCODE	Name	Operands and Conditions	
CVTTS	Convert Timestamp	Result TS	CHARV
		Source TS	CHAR
		Template	SPCPTR
DECD	Decrement Date	Result date	CHARV
		Source date	CHAR
		Duration	PVD
		Template	SPCPTR
DECT	Decrement Time	Result time	CHARV
		Source time	CHAR
		Duration	PVD
		Template	SPCPTR
DECTS	Decrement Timestamp	Result TS	CHARV
		Source TS	CHAR
		Duration	PVD
		Template	SPCPTR
INCD	Increment Date	Result date	CHARV
		Source date	CHAR
		Duration	PVD
		Template	SPCPTR
INCT	Increment Time	Result time	CHARV
		Source time	CHAR
		Duration	PVD
		Template	SPCPTR
INCTS	Increment Timestamp	Result TS	CHARV
		Source TS	CHAR
		Duration	PVD
		Template	SPCPTR

## Appendix B

### Object Type/Subtypes

#### Object Type/Subtypes

An object is identified by a 32-character string (possibly within a context) consisting of a 2-character type and subtype designation followed by a 30-character name padded out with blanks. The type and subtype are one character each and are given in hexadecimal notation, e.g. x'01EF', meaning a type of x'01' and a subtype of x'EF'.

The subtype falls in one of three categories:

- External Objects      ('01' <= Subtype Code <= '4F')
- Design Objects        ('50' <= Subtype Code <= '9F')
- Internal Objects      ('A0' <= Subtype Code <= 'FF')

#### Object List

01 Access Group		
Type/Subtype	OS/400 Object Type	Description
0190	*QDAG	Access group, composite piece
01EF	*QTAG	Temporary access group

02 Program		
Type/Subtype	OS/400 Object Type	Description
0201	*PGM	Program
0202	*SQLPKG	Structured query language package
0203	*SRVPGM	Service program
0250	*JVAPGM	Java program

03 Module		
Type/Subtype	OS/400 Object Type	Description
0301	*MODULE	Module

04 Context		
Type/Subtype	OS/400 Object Type	Description
0401	*LIB	Library
04C1	*INTLIB	Internal QTEMP library

06 Byte String Space		
Type/Subtype	OS/400 Object Type	Description
06A0	*OPTBSS	Optical byte string space
06C1	*DOCBSS	Document byte string space

07 Journal Space		
Type/Subtype	OS/400 Object Type	Description
0701	*JRNRCV	Journal receiver
07C1	*DFTRCV	Default journal receiver

08 User Profile		
Type/Subtype	OS/400 Object Type	Description
0801	*USRPRF	User profile

09 Journal Port		
Type/Subtype	OS/400 Object Type	Description
0901	*JRN	Journal
09C1	*DFTJRN	Default journal



OA Queue	
Type/Subtype	OS/400 Object Type Description
OA01	*DTAQ Data queue
OA02	*USRQ User queue
OA90	*QDQ Queue, composite piece
OAC1	*JTMMQ Measurement message queue
OAC2	*SIQ FM queue
OAC3	*DRQ Distribution recipient queue
OAC4	*DCTQ Data dictionary queue
OAC5	*DCXMSQ DCX operator message queue
OAC6	*HPQ Office/400 host print queue
OAC7	*TCP/IPQ TCP/IP internal queue
OAC8	*GENQ Generic queue
OAEF	*QTQ Temporary queue
OAF0	*JSQ Job schedule queue
OAF1	*SMQ Systems management internal queue
OAF2	*TNIPLMQ TN IPL message queue

OB Data Space	
Type/Subtype	OS/400 Object Type Description
OB50	*DIRDS Directory data space
OB51	*DEADS Directory extended attribute DS
OB90	*QDDQ Data space, composite piece
OBA0	*FIDTBL Address tables
OB5F	*QTDS Temporary data space

OC Data Space Index	
Type/Subtype	OS/400 Object Type Description
OC01	*DIR FDFS directory
OC50	*DEADI Directory extended attribute index
OC90	*QDDSI Data space index, composite piece
OCEF	*QTDSI Temporary, data space index

OD Cursor	
Type/Subtype	OS/400 Object Type Description
OD50	*MEM Database file member
OD51	*DIRCR Directory cursor
OD52	*DEACR Directory extended attribute cursor
ODEF	*OCUR Database operational cursor

OE Index	
Type/Subtype	OS/400 Object Type Description
OE01	*JOBQ Job queue
OE02	*OUTQ Output queue
OE03	*MSGF Message file
OE04	*FCT Forms control table
OE05	*SSND Session description
OE06	*IGDCT Ideographic dictionary DBCS
OE07	*SCHIDX Information search index
OE08	*RCT Reference code translate table
OE09	*ALRTBL Alert table, index
OE0A	*USRIDX User index
OE0B	*FTR Generic filter
OE0C	*JOBSCD Job schedule
OE0D	*CFGL Configuration list: V2R2
OE0E	*NODL Node list (SNA)
OE0F	*CRQD Change request description
OE10	*VLDL Validation list
OE50	*IMPLREP Implementation Repository
OE90	*QDIDX Index, composite piece
OE91	*MSRVI Master service index
OEAO	*CMTCDRI Commit recovery block
OEAI	*ZMF1NX Message space pool (Mail Server Framework)
OEAI	*TOKTBL DSOM Token Mapping Table
OEAI	*QFSIDX File Server I/O Index (FSIOP/IPC/S/INFS/etc)
OEAI	*GENIDX Generic index (incl. Job index)
OEAI	*CRGM Cluster Resource Groups
OEAI	*MNI NX Menu index
OEAI	*SDQ SNADS distribution queue
OEAI	*SECOBJ Internal Security Object
OEAI	*INTPRF Interactive profile
OEAI	*AUT Authorized user table

0EC6	*FACB	File available control block
0EC7	*PRTQ	Print queue
0EC8	*SRMDX	System resource manager index
0EC9	*SLFSMS	Secondary logic Finite State Machine index
0ECA	*FCNUL	Management Central data
0ECB	*PRODT	Operand description table
0ECC	*S36IDX	S/36 index, index
0ECD	*SWFL	System-wide folder list
0ECE	*S36HLP	S/36 help object
0ECF	*DCXITC	DSX inter task communication index
0ED0	*EDTIDX	Element description index
0ED1	*DRX	Distribution recipient index
0ED2	*CCSID1	Coded character set identifier
0EEF	*QTIDX	Temporary, index
0EF0	*X4Q	X400 queue
0EF1	*PRDAVLI	Product availability index
0EF2	*SMIDX	System management index

<b>0F Commit Block</b>		
Type/Subtype	OS/400 Object Type	Description
0FC1	*CBLK	Commit block

<b>10 Logical Unit Description</b>		
Type/Subtype	OS/400 Object Type	Description
1001	*DEV D	Device description

<b>11 Network Description</b>		
Type/Subtype	OS/400 Object Type	Description
1101	*LIND	Line description

<b>12 Controller Description</b>		
Type/Subtype	OS/400 Object Type	Description
1201	*CTLD	Control unit description

<b>13 Dump Space</b>		
Type/Subtype	OS/400 Object Type	Description
1390	*DMPSP	Dump space

<b>14 Service Description</b>		
Type/Subtype	OS/400 Object Type	Description
1401	*COSD	Class of service description

<b>15 Mode Description</b>		
Type/Subtype	OS/400 Object Type	Description
1501	*MODD	Mode description

<b>16 Network Interface</b>		
Type/Subtype	OS/400 Object Type	Description
1601	*NWI D	Network interface description

<b>17 Connection List</b>		
Type/Subtype	OS/400 Object Type	Description
1701	*CNNL	Connection list

<b>18 Queue Space</b>		
Type/Subtype	OS/400 Object Type	Description
18A0	*JMQ	Job message queue
18A1	*IPLJMQ	VMC internally created queue space

<b>19 Space</b>		
Type/Subtype	OS/400 Object Type	Description
1900	*SP	Space
1901	*FILE	File
1902	*MSGQ	Message queue
1903	*JOB D	Job description
1904	*CLS	Class
1905	*CMD	Command
1906	*TBL	Table
1907	*PRTIMG	Print image

1908	*EDTD	Edit description
1909	*SBSD	Subsystem description
190A	*DTAARA	Data area
190B	*CLD	C locale description, space
190C	*GSS	Graphics symbol set
190D	*CHTFMT	Chart format
190E	*DOC	Document
190F	*DOCL	Document list
1910	*IGCTBL	DBCS ideographic font character table
1911	*QRYDFN	Query definition
1912	*FLR	Folder
1913	*EXITRG	Exit registration
1914	*NTBD	NetBIOS configuration data
1915	*PNLGRP	Panel group definition
1916	*MENU	Menu definition
1917	*SVRSTG	Data storage for IPCS (or FSIOP)
1918	*CFGLO	Configuration list: Pre V2R2
1919	*S36	S/36 environment description
191A	*IGCSRT	Ideographic sort table
191B	*PRDDFN	Product definition
191C	*PRDFUN	Product function
191D	*PRDLOD	Product load
191E	*IPXD	IPX Description data (From CRTIPXD)
191F	*SOLUDT	SQL user defined type
1920	*DTADCT	Data dictionary
1921	*LOCALE	"C" Locale information
1922	*CSPMAP	Cross system product map
1923	*CSPTBL	Cross system product table, space
1924	*M36CFG	Advanced System/36 configuration
1925	*PSFCFG	Print Services Facility Configuration
1926	*FNTRSC	Font resource, space
1927	*PAGSEG	Page segment, space
1928	*FORMDF	Form definition, space
1929	*OVL	Overlay, space
192A	*NODGRP	Node group
192B	*FNTTBL	Font definition Table
192C	*CRG	Cluster resource group
192D	*MGTCOL	Management collection
1930	*PDG	Print descriptor group
1931	*QMORY	Query management query
1932	*QMFORM	Query management form
1933	*PRDAVL	Product availability
1934	*USRSPC	User space
1935	*CSI	Communications side information
1936	*PAGDFN	Page definition
1937	*BNDDIR	Binding directory
1938	*WSCST	Work station customizing object
1950	*DBDIR	Database Directory Database dictionary
1951	*FMT	File format
1952	*OIRS	OIR composite piece
1953	*EXITSP	Exit registration space
1954	*SVRSTGD	Server storage description
1955	*DBCOLES	??
1958	*FCS	File constraint extension to FDT
1959	*GRPDLS	??
1990	*QDSP	Space, composite piece
19A0	*RCYAP	Recovery times for SMAPP
19A1	*PRMGEN	Permanent generic space
19A2	*ZMFSPC	Mail Server Message framework space
19A3	*RZHRI PD	Hardware Resources persistent data
19A4	*CFGSPC	Configuration space
19A5	*CIO	??
19C0	*MCO	Measurement collection object
19C1	*INITSP	Install initial template space
19C2	*SPLCB	Spool control block
19C3	*SEPT	System entry point table
19C5	*RWCB	Read/write control block
19C6	*ICO	Install communication object
19C7	*PDT	Process definition template
19C8	*MCOTBL	Measurement collection obj table
19C9	*MDO	Measurement descriptor object
19CA	*JAR	Job APAR repository
19CB	*MNTXT	Menu text

19CC	*PCCR	Program change control record
19CD	*GDA	Group data area
19CE	*LDA	Local data area
19CF	*SROUTH	Save/restore authorizations table
19D0	*WCBT	Work control block table
19D1	*LIBRCVR	Library recovery object
19D2	*SVAL	System value space object (QWMSYSVAL)
19D3	*SYSCB	System control block
19D4	*DBRCVR	Database recovery object
19D5	*I NAUT	Install authority object
19D6	*SYSPTI	System print image part numbers
19D7	*EPTAB	Data management entry point table
19D8	*SYSRPYL	System reply list
19D9	*UFCB	User file control block
19DA	*PROCT	MI operation code table
19DB	*SRDS	Save/restore descriptor space
19DC	*SCO	Service communication object
19DD	*WCBTRO	WCBT recovery object
19DE	*SCPFSP	SCPF space
19DF	*MQLOCK	Message queue locking protocol
19E0	*ADO	Asynchronous distribution object
19E1	*IGCINT	Ideographic character table
19E2	*DTO	Distribution tracking object
19E3	*DUO	Document unit object
19E4	*OSSCB	Session control block (Document Interchange Architecture)
19E5	*NFSP	Network facility space
19E6	*MDOC	Mail document
19E7	*UFO	Un-filed folder object
19E8	*FSO	FMS system object
19E9	*DSNXO	DSNX system object
19EA	*S36EPT	S/36 entry point table
19EB	*IDDEDT	Internal data dictionary
19EC	*S36HST	S/36 history object
19ED	*VMCLSF	VLIC LSF holder
19EE	*MSCSP	Permanent miscellaneous space
19EF	*QTSP	Temporary space
19F0	*ACNAME	Automatic configuration names
19F1	*S36BCH	S/36 batch object
19F2	*LIBRCVR	Library REC object for rename
19F3	*SRMSPC	System resource manager space
19F4	*I NAUTO	Auto install object
19F5	*DCRENO	DC rename object
19F6	*SYAUTS	Security authorization space
19F7	*HFSO	HFS description
19F8	*SHRCV	SH recovery object
19F9	*SNMTBL	System program name table
19FA	*X40	X400 object
19FB	*CNVTBL	System conversion table
19FC	*PTCSPC	Protected space
19FE	*UBPSPC	Usage based pricing space

1A Process Control Space		
Type/Subtype	OS/400 Object Type	Description
1A90	*QDPCS	Process Control Space, composite piece
1AEF	*QTPCS	Temporary process control space

1B Authorization List		
Type/Subtype	OS/400 Object Type	Description
1B01	*AUTL	Authorization list
1BC1	*AUTHLR	Authority holder

1C Spelling Aid Dictionary		
Type/Subtype	OS/400 Object Type	Description
1C01	*SPADCT	Spelling aid dictionary

1D Network Description		
Type/Subtype	OS/400 Object Type	Description
1D01	*NWSD	Network server description

<b>1E Stream Files</b>		
Type/Subtype	OS/400 Object Type	Description
1E01	*STMF	Stream file
1E02	*SYMLNK	Symbolic link
1E03	*SOCKET	Local socket
1E04	*M36	Advanced System/36 Machine description
1E05	*BLKSF	Block special file
1E52	*MCBSF	MCB Special File
1EA0	*DLSTMF	Document library stream file
1EB1	*SMBSF	SMB special file (Samba)
1EED	*OPTSTMF	Optical stream file

<b>1F Distributed Files</b>		
Type/Subtype	OS/400 Object Type	Description
1F01	*DSTMF	Distributed stream file
1F02	*DDIR	Distributes directory

<b>20 System Object Model</b>		
Type/Subtype	OS/400 Object Type	Description
2001	*SOMOBJ	System object model
2002	*OOP00L	??

<b>21 Composite Object Groups</b>		
Type/Subtype	OS/400 Object Type	Description
2150	*JVAGRP	Java group

<b>81 Machine Context</b>		
Type/Subtype	OS/400 Object Type	Description
8100	*MCHCTX	Machine context

<b>85 Stream Object</b>		
Type/Subtype	OS/400 Object Type	Description
85A0	*STREAM	Stream object

## Appendix C

### PowerPC Instruction Set Quick Reference

#### Instructions Sorted by Mnemonic

The following table lists the instructions for the AS/400 version of the PowerPC in alphabetic order by mnemonic operation code. Instructions marked with a little ring like “ADD<sub>o</sub>” can record the result in the condition register if you write a period or “dot” after the mnemonic, like “ADD.”. These instructions also have an “Rc” field in bit position 31. Instructions that already have a period after the mnemonic are always recording, like **ADDIC.**. Privileged instructions that require supervisor mode are shown in *red*:

Name	0	5	6	10	11	15	16	20	21	30	31
ADD <sub>o</sub>	31		D		A		B		OE	266	Rc
ADD <sub>o</sub> C.	31		D		A		B		OE	10	Rc
ADDE <sub>o</sub>	31		D		A		B		OE	138	Rc
ADDI	14		D		A					SIMM	
ADDIC	12		D		A					SIMM	
ADDIC.	13		D		A					SIMM	
ADDIS	15		D		A					SIMM	
ADDME <sub>o</sub>	31		D		A		00000		OE	234	Rc
ADDZE <sub>o</sub>	31		D		A		00000		OE	202	Rc
AND <sub>o</sub>	31		S		A		B			28	Rc
ANDC <sub>o</sub>	31		S		A		B			60	Rc
ANDI.	28		S		A					UIMM	
ANDIS.	29		S		A					UIMM	
B	18						LI			0	0
BL	18						LI			0	1
BA	18						LI			1	0
BLA	18						LI			1	1
BC	16		BO		BI			BD		0	0
BCL	16		BO		BI			BD		0	1
BCA	16		BO		BI			BD		1	0
BCLA	16		BO		BI			BD		1	1
BCCTR	19		BO		BI		00000			528	0
BCCTRL	19		BO		BI		00000			528	1
BCLR	19		BO		BI		00000			16	0
BCLRL	19		BO		BI		00000			16	1
CMP	31		CD	0	L	A	B			0	0
CMPI	11		CD	0	L	A				SIMM	
CMPL	31		CD	0	L	A	B			32	0
CMPLI	10		CD	0	L	A				UIMM	
CNTIZD <sub>o</sub>	31		S		A		00000			58	Rc
CNTIZW <sub>o</sub>	31		S		A		00000			26	Rc
CRAND	19		CD		CA		CB			257	0
CRANDC	19		CD		CA		CB			129	0
CREQV	19		CD		CA		CB			289	0
CRNAND	19		CD		CA		CB			225	0
CRNOR	19		CD		CA		CB			33	0
CROR	19		CD		CA		CB			449	0
CRORC	19		CD		CA		CB			417	0
CRXOR	19		CD		CA		CB			193	0
DCBA	31		00000		A		B			758	0
DCBF	31		00000		A		B			86	0
<i>DCBI</i>	31		00000		A		B			470	0
DCBST	31		00000		A		B			54	0
DCBT	31		00000		A		B			278	0
DCBTST	31		00000		A		B			246	0
DCBZ	31		00000		A		B			1014	0
DIVD <sub>o</sub>	31		D		A		B		OE	489	Rc
DIVDU <sub>o</sub>	31		D		A		B		OE	457	Rc
DIVW <sub>o</sub>	31		D		A		B		OE	491	Rc
DIVWU <sub>o</sub>	31		D		A		B		OE	458	Rc
ECIWUX	31		D		A		B			310	0
ECOWUX	31		S		A		B			438	0
EIEIO	31		00000		00000		00000			854	0
EQV	31		S		A		B			284	Rc
EXTSB <sub>o</sub>	31		S		A		00000			954	Rc
EXTSH <sub>o</sub>	31		S		A		00000			922	Rc
EXTSB <sub>o</sub>	31		S		A		00000			985	Rc

FABS <sub>o</sub>	63	D	00000	B	264	Rc
FADD <sub>o</sub>	63	D	A	B	00000 21	Rc
FADDS <sub>o</sub>	59	D	A	B	00000 21	Rc
FCFID <sub>o</sub>	63	D	00000	B	846	Rc
FCMPO <sub>o</sub>	63	CD 00	A	B	32	0
FCMPU <sub>o</sub>	63	CD 00	A	B	0	0
FCTID <sub>o</sub>	63	D	00000	B	814	Rc
FCTIDZ <sub>o</sub>	63	D	00000	B	815	Rc
FCTIW <sub>o</sub>	63	D	00000	B	14	Rc
FCTIWZ <sub>o</sub>	63	D	00000	B	15	Rc
FDIV <sub>o</sub>	63	D	A	B	00000 18	Rc
FDIVS <sub>o</sub>	59	D	A	B	00000 18	Rc
FMADD <sub>o</sub>	63	D	A	B	C 29	Rc
FMADDS <sub>o</sub>	59	D	A	B	C 29	Rc
FMR <sub>o</sub>	63	D	00000	B	72	Rc
FMSUB <sub>o</sub>	63	D	A	B	C 28	Rc
FMSUBS <sub>o</sub>	59	D	A	B	C 28	Rc
FMUL <sub>o</sub>	63	D	A	00000	C 25	Rc
FMULS <sub>o</sub>	59	D	A	00000	C 25	Rc
FNABS <sub>o</sub>	63	D	00000	B	136	Rc
FNEG <sub>o</sub>	63	D	00000	B	40	Rc
FNMADD <sub>o</sub>	63	D	A	B	C 31	Rc
FNMADDS <sub>o</sub>	59	D	A	B	C 31	Rc
FNMSUB <sub>o</sub>	63	D	A	B	C 30	Rc
FNMSUBS <sub>o</sub>	59	D	A	B	C 30	Rc
FRES <sub>o</sub>	59	D	00000	B	00000 24	Rc
FRSP <sub>o</sub>	63	D	00000	B	12	Rc
FRSQRT <sub>o</sub>	63	D	00000	B	00000 26	Rc
FSEL <sub>o</sub>	63	D	A	B	C 23	Rc
FSQRT <sub>o</sub>	63	D	00000	B	00000 22	Rc
FSQRTS <sub>o</sub>	59	D	00000	B	00000 22	Rc
FSUB <sub>o</sub>	63	D	A	B	00000 20	Rc
FSUBS <sub>o</sub>	59	D	A	B	00000 20	Rc
ICBI	31	00000	A	B	982	0
ISYNC	19	00000	00000	00000	150	0
LBZ	34	D	A	d		
LBZU	35	D	A	d		
LBZUX	31	D	A	B	119	0
LBZX	31	D	A	B	87	0
LD	58	D	A	ds		0
LDARX	31	D	A	B	84	0
LDU	58	D	A	ds		1
LDUX	31	D	A	B	53	0
LDX	31	D	A	B	21	0
LFD	50	D	A	d		
LFDU	51	D	A	d		
LFDUX	31	D	A	B	631	0
LFDX	31	D	A	B	599	0
LFS	48	D	A	d		
LFSU	49	D	A	d		
LFSUX	31	D	A	B	567	0
LFSX	31	D	A	B	535	0
LHA	48	D	A	d		
LHAU	49	D	A	d		
LHAUX	31	D	A	B	375	0
LHAX	31	D	A	B	343	0
LHBRX	31	D	A	B	790	0
LHZ	40	D	A	d		
LHZU	41	D	A	d		
LHZUX	31	D	A	B	311	0
LHZX	31	D	A	B	279	0
LMW	46	D	A	d		
LMD	58	D	A	ds		3
LQ	56	D	A	ds	tag	
LSDI	31	D	A	B	629	0
LSDX	31	D	A	B	565	0
LSWI	31	D	A	B	597	0
LSWX	31	D	A	B	533	0
LWA	58	D	A	ds		2
LWARX	31	D	A	B	20	0
LWAUX	31	D	A	B	373	0
LWAX	31	D	A	B	341	0
LWBRX	31	D	A	B	534	0
LWZ	32	D	A	d		
LWZU	33	D	A	d		

LWZUX	31	D		A	B	55	0
LWZX	31	D		A	B	23	0
MCRF	31	CD	00	CS	00	00000	0
MCRFS	31	CD	00	CS	00	00000	64
MCRXR	31	CD	00	00000	00000	512	0
MFCR	31	D		00000	00000	19	0
MFFS	63	D		00000	00000	19	Rc
MFMSR	31	D		00000	00000	83	0
MFSPR	31	D		spr		339	0
MFSSR	31	D	0	SR	00000	595	0
MFMSRIN	31	D		00000	B	659	0
MFTB	31	D		tbr		371	0
MTCRF	31	S	0	CRM	0	144	0
MTFSB0	63	CD		00000	00000	70	Rc
MTFSB1	63	CD		00000	00000	38	Rc
MTFSF	63	0	FM	0	00000	711	Rc
MTFSFI	63	CD	00	00000	IMM	134	Rc
MTMSR	31	S		00000	00000	146	0
MTMSRD	31	S		00000	00000	178	0
MTSPR	31	S		spr		467	0
MTSR	31	S	0	SR	00000	210	0
MTSRD	31	S	0	SR	00000	82	0
MTSRDIN	31	S		00000	B	114	0
MTSRIN	31	S		00000	B	242	0
MULHD	31	D		A	B	0	Rc
MULHDU	31	D		A	B	0	Rc
MULHW	31	D		A	B	0	Rc
MULHWU	31	D		A	B	0	Rc
MULLD	31	D		A	B	OE	Rc
MULLI	7	D		A		SIMM	
MULLW	31	D		A	B	OE	Rc
NAND	31	S		A	B	476	Rc
NEG	31	D		A	00000	OE	Rc
NOR	31	S		A	B	124	Rc
OR	31	S		A	B	444	Rc
ORC	31	S		A	B	412	Rc
ORI	24	S		A		UIMM	
ORIS	25	S		A		UIMM	
RFI	19	00000		00000	00000	50	0
RFID	19	00000		00000	00000	18	0
RFSCV	19	00000		00000	00000	82	0
RLDICL	30	S		A	B	mb	Rc
RLDICR	30	S		A	B	mb	Rc
RLDIC	30	S		A	sh	mb	Rc
RLDICL	30	S		A	sh	mb	Rc
RLDICR	30	S		A	sh	mb	Rc
RLDIMI	30	S		A	sh	mb	Rc
RLWIMI	20	S		A	SH	MB	Rc
RLWINM	21	S		A	SH	MB	Rc
RLWNM	23	S		A	B	MB	Rc
SC	17	00000		00000	00000	0000000000000000	0
SCV	17	00000		00000	00000	SCVn	0
SETTAG	31	S		A	00000	499	0
SLBIA	31	00000		00000	00000	498	0
SLBIE	31	00000		00000	B	434	0
SLD	31	S		A	B	27	Rc
SLW	31	S		A	B	24	Rc
SRAD	31	S		A	B	794	Rc
SRADI	31	S		A	sh	413	Rc
SRAW	31	S		A	B	792	Rc
SRAWI	31	S		A	B	824	Rc
SRD	31	S		A	B	539	Rc
SRW	31	S		A	SH	536	Rc
STB	38	S		A		d	
STBU	39	S		A		d	
STBUX	31	S		A	B	247	0
STBX	31	S		A	B	215	0
STD	62	S		A		ds	0
STDCX	31	S		A	B	214	1
STDU	62	S		A		ds	1
STDUX	31	S		A	B	181	0
STDY	31	S		A	B	149	0
STFD	54	S		A		d	
STFDU	55	S		A		d	
STFDUX	31	S		A	B	759	0



STFDX	31	S	A	B	727	0
STFIWX	31	S	A	B	983	0
STFS	52	S	A	d		
STFSU	53	S	A	d		
STFSUX	31	S	A	B	695	0
STFSX	31	S	A	B	663	0
STH	44	S	A	d		
STHBRX	31	S	A	B	918	0
STHU	45	S	A	d		
STHUX	31	S	A	B	439	0
STHX	31	S	A	B	407	0
STMW	45	S	A	d		
STMD	62	S	A	ds		3
STQ	62	S	A	ds		2
STSDI	31	S	A	B	757	0
STSDX	31	S	A	B	693	0
STSWX	31	S	A	B	661	0
STSWI	31	S	A	B	725	0
STSWX	31	S	A	B	661	0
STW	36	S	A	d		
STWBRX	31	S	A	B	662	0
STWCX.	31	S	A	B	150	1
STWU	36	S	A	d		
STWUX	31	S	A	B	183	0
STWX	31	S	A	B	151	0
SUBF.	31	D	A	00000	OE	40
SUBFC.	31	D	A	00000	OE	8
SUBFE.	31	D	A	00000	OE	136
SUBFIC	8	D	A	SIMM		
SUBFNE.	31	D	A	00000	OE	232
SUBFZE.	31	D	A	00000	OE	200
SYNC	31	00000	00000	00000	598	0
TD	31	TO	A	B	68	0
TDIC	2	TO	A	SIMM		
TLBIA	31	00000	00000	00000	379	0
TLBIE	31	00000	00000	B	306	0
TLBSYNC	31	00000	00000	00000	566	0
TW	31	TO	A	B	68	0
TWIC	3	TO	A	SIMM		
TXER	31	TO	A	00000	TN	36
XOR.	31	S	A	B	326	Rc
XORI	26	S	A	UIMM		
XORIS	27	S	A	UIMM		

## Appendix D

### SEPT Entries in the User Domain

#### System Entry Point Table Entries

This appendix contains all SEPT entries that are in the *user domain* (can be called by user-state programs). For each program the “Authority” column specifies if the program adopts authority from the \*USER or from the \*OWNER (QSYS). The values are from a V4R4M0 system. First we show the entries sorted by number (1-based) and then by name.

#### Entries Sorted by Number

Nbr	Name	LI b	Date	Time	Authority	Size	State
0011	QDMCLOSE	QSYS	1998/12/08	09:13:27	*USER	49152	*SYSTEM
0012	QDMCOPEN	QSYS	1999/04/20	11:53:39	*USER	180224	*SYSTEM
0014	QDBGETDR	QSYS	1999/10/11	14:39:30	*USER	143360	*SYSTEM
0015	QDBGETKY	QSYS	1999/10/11	14:36:46	*USER	155648	*SYSTEM
0016	QDBGETSQ	QSYS	1999/10/11	14:35:49	*USER	151552	*SYSTEM
0018	QDBPUT	QSYS	1999/08/25	13:26:17	*USER	139264	*SYSTEM
0019	QDBUDR	QSYS	1999/08/25	13:24:58	*USER	167936	*SYSTEM
0022	QWSPUT	QSYS	2000/07/21	14:46:14	*USER	544768	*SYSTEM
0030	QBSGET	QSYS	1998/12/08	06:53:12	*USER	69632	*SYSTEM
0039	QBSPUT	QSYS	1998/12/08	06:00:54	*USER	45056	*SYSTEM
0045	QSPBPDSK	QSYS	1998/12/08	10:06:03	*USER	24576	*SYSTEM
0052	QWSGET	QSYS	2000/07/22	14:56:56	*USER	397312	*SYSTEM
0058	QTAGET	QSYS	1998/12/09	03:06:16	*USER	40960	*SYSTEM
0059	QTAPUT	QSYS	1999/04/24	10:10:49	*USER	32768	*SYSTEM
0060	QDKGET	QSYS	1998/12/08	07:50:25	*USER	28672	*SYSTEM
0061	QDKPUT	QSYS	1998/12/08	09:10:40	*USER	28672	*SYSTEM
0063	QDMNODEV	QSYS	1998/12/08	09:15:23	*USER	28672	*SYSTEM
0066	QDMACQDV	QSYS	1998/12/09	02:23:02	*USER	73728	*SYSTEM
0069	QDMI FERR	QSYS	1998/12/08	07:53:07	*USER	32768	*SYSTEM
0071	QDMRLSDV	QSYS	1998/12/08	09:17:53	*USER	32768	*SYSTEM
0074	QLI CNV	QSYS	1998/12/08	09:53:48	*USER	28672	*SYSTEM
0081	QWCSVRDR	QSYS	1998/12/08	15:17:29	*USER	20480	*SYSTEM
0082	QWCSVRTR	QSYS	1998/12/08	17:22:55	*USER	20480	*SYSTEM
0086	QWCCRCRC	QSYS	1998/12/08	17:17:26	*USER	24576	*SYSTEM
0092	QTAFEO	QSYS	1999/04/24	10:09:47	*USER	28672	*SYSTEM
0095	QLPBATI N	QSYS	1999/02/18	22:09:13	*OWNER	77824	*SYSTEM
0097	QDKFEO	QSYS	1998/12/08	09:10:31	*USER	24576	*SYSTEM
0107	QWPFEOD	QSYS	1998/12/08	11:34:46	*USER	40960	*SYSTEM
0111	QDBFEO	QSYS	1999/01/30	20:15:35	*USER	90112	*SYSTEM
0112	QSPFEOD	QSYS	1998/12/22	13:03:28	*OWNER	106496	*SYSTEM
0121	QDMACCI N	QSYS	1998/12/08	06:54:44	*USER	32768	*SYSTEM
0194	QSPRDR	QSYS	1998/12/08	13:19:44	*USER	102400	*SYSTEM
0195	QSPWTRM1	QSYS	1999/01/30	21:14:47	*OWNER	65536	*SYSTEM
0259	QCL	QSYS	1998/12/08	07:11:11	*USER	57344	*SYSTEM
0276	QCAEXEC	QSYS	1998/12/08	06:54:32	*USER	32768	*SYSTEM
0277	QCADRV	QSYS	1998/12/08	06:01:36	*USER	40960	*SYSTEM
0309	QPGMMENU	QSYS	1998/12/08	12:08:18	*USER	143360	*SYSTEM
0312	QCLCLCPR	QSYS	1998/12/08	07:12:11	*USER	61440	*SYSTEM
0313	QCLRTNE	QSYS	1998/12/08	06:13:13	*USER	20480	*SYSTEM
0314	QCLCLNUP	QSYS	1998/12/08	06:17:13	*USER	28672	*SYSTEM
0315	QCLDMI O	QSYS	1999/04/20	11:51:43	*USER	53248	*SYSTEM
0316	QCLRSLV	QSYS	1998/12/08	07:18:26	*USER	49152	*SYSTEM
0317	QCLRSVRE	QSYS	1998/12/08	06:20:38	*USER	24576	*SYSTEM
0318	QCLSSVRE	QSYS	1998/12/08	07:20:18	*USER	28672	*SYSTEM
0319	QCLXCExc	QSYS	1998/12/08	06:22:30	*USER	20480	*SYSTEM
0334	QCLXERR	QSYS	1998/12/08	06:14:24	*USER	24576	*SYSTEM
0373	QPOLFFDC	QSYS	1998/12/14	02:41:26	*USER	40960	*INHERIT
0410	QDCXLATE	QSYS	1998/12/08	08:57:08	*USER	61440	*SYSTEM
0411	QECEDI TW	QSYS	1998/12/08	09:28:16	*USER	28672	*INHERIT
0701	QYYGETK	QSYS	1998/12/09	04:49:00	*USER	61440	*SYSTEM
0757	QEARBKM	QSYS	1998/12/08	06:59:05	*USER	24576	*SYSTEM
0768	QDBPUTM	QSYS	1999/01/30	20:23:19	*USER	90112	*SYSTEM
0770	QDBGETM	QSYS	1999/10/18	10:57:04	*USER	196608	*SYSTEM
0787	QTAFEOV	QSYS	1999/04/24	10:09:49	*USER	24576	*SYSTEM

Nbr	Name	Li b	Date	Time	Authori ty	Size	State
0795	QSLPUT	QSYS	1998/12/08	13: 04: 17	*USER	86016	*SYSTEM
0796	QSLGET	QSYS	1998/12/08	13: 04: 07	*USER	81920	*SYSTEM
0887	QCLCNVCN	QSYS	1998/12/08	07: 10: 57	*USER	24576	*I NHERI T
0888	QCLCNVNC	QSYS	1998/12/08	06: 17: 53	*USER	20480	*I NHERI T
0889	QCLSWTCH	QSYS	1998/12/08	06: 14: 15	*USER	20480	*I NHERI T
0946	QDKPUTI	QSYS	1998/12/08	07: 50: 22	*USER	28672	*SYSTEM
0947	QDKGETI	QSYS	1998/12/08	07: 50: 02	*USER	28672	*SYSTEM
1032	QSI GET	QSYS	1998/12/08	15: 26: 00	*USER	49152	*SYSTEM
1033	QSI PUT	QSYS	1998/12/08	10: 02: 22	*USER	65536	*SYSTEM
1101	QYYCGETD	QSYS	1998/12/09	03: 36: 34	*USER	61440	*SYSTEM
1102	QTNROLLB	QSYS	1999/08/09	18: 52: 31	*OWNER	139264	*SYSTEM
1103	QTNCOMT	QSYS	1999/07/29	19: 09: 03	*OWNER	86016	*SYSTEM
1158	QWSSETWS	QSYS	1998/12/08	15: 28: 21	*USER	24576	*SYSTEM
1196	QEMPEBSC	QSYS	1999/05/25	14: 51: 11	*USER	69632	*SYSTEM
1197	QEMPESNA	QSYS	1999/05/25	14: 51: 50	*USER	102400	*SYSTEM
1232	QWPGGRAPH	QSYS	1998/12/08	17: 26: 48	*USER	28672	*SYSTEM
1244	QSRFEOD	QSYS	1998/12/08	15: 54: 52	*USER	24576	*SYSTEM
1246	QSRGET	QSYS	1998/12/08	10: 19: 02	*USER	24576	*SYSTEM
1248	QSRPUT	QSYS	1998/12/08	15: 56: 12	*USER	24576	*SYSTEM
1255	QFNROUTE	QSYS	1998/12/08	10: 18: 03	*USER	65536	*SYSTEM
1257	QFNREAD	QSYS	1998/12/08	07: 18: 40	*USER	20480	*SYSTEM
1258	QFNWRT	QSYS	1998/12/08	08: 45: 29	*USER	20480	*SYSTEM
1259	QFNWRTI	QSYS	1998/12/08	07: 19: 10	*USER	20480	*SYSTEM
1265	QGDACO	QSYS	1998/12/08	10: 31: 59	*USER	57344	*SYSTEM
1267	QGDAL NVP	QSYS	1998/12/08	08: 58: 36	*USER	20480	*SYSTEM
1418	QCOEXI T	QSYS	1998/12/12	04: 26: 18	*USER	28672	*USER
1540	QWPI PTFL	QSYS	1999/09/01	13: 15: 41	*USER	126976	*SYSTEM
1544	QWSPUDDS	QSYS	2000/04/12	17: 14: 41	*USER	94208	*SYSTEM
1550	QFNREADI	QSYS	1998/12/08	10: 15: 48	*USER	20480	*SYSTEM
1604	QWSQRYWS	QSYS	1998/12/08	17: 30: 52	*USER	24576	*SYSTEM
1605	QOSPRINT	QSYS	1998/12/08	11: 48: 27	*OWNER	24576	*SYSTEM
1610	QWPI PUT	QSYS	1998/12/08	11: 37: 16	*USER	86016	*SYSTEM
1621	QTSUDR	QSYS	1998/12/08	11: 07: 47	*USER	45056	*SYSTEM
1623	QTSPUT	QSYS	1998/12/08	16: 54: 08	*USER	69632	*SYSTEM
1624	QTSPUTM	QSYS	1998/12/08	14: 37: 25	*USER	57344	*SYSTEM
1631	QTSFEOD	QSYS	1998/12/08	16: 52: 12	*USER	77824	*SYSTEM
1632	QTSGET	QSYS	1998/12/09	03: 23: 11	*USER	61440	*SYSTEM
1633	QTSGETM	QSYS	1998/12/08	14: 35: 55	*USER	53248	*SYSTEM
1665	QCI OPEN	QSYS	1998/12/08	07: 03: 31	*USER	24576	*SYSTEM
1666	QCI MRT	QSYS	1998/12/08	07: 06: 25	*USER	118784	*SYSTEM
1667	QCI ACCI N	QSYS	1998/12/08	06: 08: 14	*USER	57344	*SYSTEM
1678	QDBPUTD	QSYS	1998/12/22	13: 38: 29	*USER	32768	*SYSTEM
1685	QTSGETSQ	QSYS	1998/12/08	14: 35: 20	*USER	20480	*SYSTEM
1698	QI CGET	QSYS	1999/01/07	14: 33: 16	*USER	73728	*SYSTEM
1699	QI CPUT	QSYS	1998/12/08	09: 17: 50	*USER	151552	*SYSTEM
1703	QY2FTML	QSYS	1998/12/09	03: 30: 00	*USER	151552	*SYSTEM
1712	QDBPUTDR	QSYS	1999/08/25	13: 25: 37	*USER	139264	*SYSTEM
1819	QWPLNPR	QSYS	1999/05/07	10: 05: 56	*USER	73728	*SYSTEM
1852	QUSCMDLN	QSYS	1998/12/08	17: 10: 00	*USER	24576	*SYSTEM
1876	SUBRF1	QSYS	1998/11/18	02: 52: 50	*USER	20480	*USER
1877	SUBRF2	QSYS	1998/11/17	22: 19: 18	*USER	20480	*USER
1878	SUBRA1	QSYS	1998/12/08	17: 55: 57	*USER	24576	*SYSTEM
1885	QWPASPR	QSYS	1998/12/08	17: 26: 28	*USER	28672	*SYSTEM
1900	QEZWRCLU	QSYS	1998/12/08	08: 41: 31	*OWNER	32768	*SYSTEM
1929	QSZRECOV	QSYS	1999/01/29	23: 55: 04	*OWNER	45056	*SYSTEM
1951	QDMGETST	QSYS	1998/12/08	09: 17: 17	*USER	57344	*SYSTEM
1953	QWPPUT	QSYS	1998/12/08	17: 29: 37	*USER	106496	*SYSTEM
1954	QWPPFTFLD	QSYS	1999/09/01	13: 14: 46	*USER	122880	*SYSTEM
1962	QCMD	QSYS	1998/12/08	07: 25: 55	*USER	81920	*SYSTEM
1988	QCMDCHK	QSYS	1998/12/08	06: 22: 25	*USER	36864	*SYSTEM
1989	QCMDEXC	QSYS	1998/12/08	06: 23: 20	*USER	32768	*SYSTEM
2057	QSCCPY	QSYS	1998/12/08	12: 54: 27	*OWNER	49152	*SYSTEM
2127	QCI GETUL	QSYS	1998/12/08	06: 59: 52	*USER	20480	*SYSTEM
2180	QOHFI XI X	QSYS	1998/12/08	13: 23: 55	*OWNER	20480	*SYSTEM
2181	QOLELI NK	QSYS	1998/12/08	11: 25: 57	*OWNER	73728	*SYSTEM
2182	QOLDLI NK	QSYS	1998/12/08	13: 37: 39	*USER	45056	*SYSTEM
2186	QOLSEND	QSYS	1998/12/08	11: 23: 58	*OWNER	45056	*SYSTEM
2187	QOLRECV	QSYS	1998/12/08	11: 23: 46	*OWNER	40960	*SYSTEM
2188	QOLSETF	QSYS	1998/12/08	13: 37: 57	*USER	32768	*SYSTEM
2189	QOLQLI ND	QSYS	1998/12/08	13: 39: 23	*USER	40960	*SYSTEM
2211	QOLTI MER	QSYS	1998/12/08	13: 38: 22	*USER	28672	*SYSTEM
2284	QTSPUTDR	QSYS	1998/12/08	16: 54: 01	*USER	49152	*SYSTEM

Nbr	Name	Li b	Date	Time	Authori ty	Size	State
2357	QMNSBS	QSYS	1998/12/08	08: 15: 59	*USER	20480	*SYSTEM
2358	QMNUI M	QSYS	1999/08/01	17: 22: 26	*USER	36864	*SYSTEM
2421	QOECMPRT	QSYS	1999/06/04	13: 33: 28	*USER	36864	*SYSTEM
2445	QMHRTVRQ	QSYS	1998/12/08	12: 30: 32	*USER	36864	*SYSTEM
2488	QY11TFAT	QSYS	1998/12/08	15: 43: 40	*USER	16384	*SYSTEM
2494	QBNTGTRL	QSYS	1998/12/08	05: 57: 25	*USER	20480	*USER
2643	QSROUTE	QSYS	2000/02/17	18: 24: 14	*USER	1228800	*SYSTEM
2650	QSPSETWI	QSYS	1998/12/08	13: 20: 59	*OWNER	61440	*SYSTEM
2679	QSUWHPCP	QSYS	1998/12/08	10: 23: 05	*USER	24576	*SYSTEM
2767	QDXHRTR	QSYS	1998/12/08	09: 24: 27	*OWNER	102400	*SYSTEM
2796	QEXFPGM	QSYS	1998/12/08	08: 26: 53	*USER	24576	*SYSTEM
2823	QTBXLATE	QSYS	1998/12/08	14: 09: 04	*USER	61440	*SYSTEM
2848	QFSMAI N	QSYS	1998/12/08	10: 31: 05	*USER	24576	*SYSTEM
2856	QMNHCMD	QSYS	1998/12/08	10: 31: 50	*USER	45056	*SYSTEM
2858	QCJONCI R	QSYS	1998/12/08	07: 08: 21	*USER	28672	*USER
2878	QCJROUT	QSYS	1998/12/08	06: 17: 12	*USER	98304	*SYSTEM
2893	QEZCNPWR	QSYS	1998/12/08	08: 35: 13	*USER	28672	*SYSTEM
2894	QEZATNWD	QSYS	1998/12/08	08: 31: 36	*USER	24576	*SYSTEM
2896	QEZMAI N	QSYS	1998/12/08	10: 06: 52	*USER	20480	*SYSTEM
2897	QRCVDTAQ	QSYS	1999/04/28	12: 09: 29	*USER	53248	*SYSTEM
2898	QSNDDTAQ	QSYS	1999/04/28	12: 09: 07	*USER	40960	*SYSTEM
2899	QCLRDTAQ	QSYS	1998/12/08	06: 21: 03	*USER	40960	*SYSTEM
2926	QCLSCAN	QSYS	1998/12/08	06: 21: 21	*USER	24576	*I NHERI T
2927	QACACHEK	QSYS	1998/12/08	06: 00: 41	*USER	36864	*SYSTEM
2938	QSRDUPER	QSYS	1998/12/08	10: 19: 26	*USER	40960	*SYSTEM
2987	QZMFXTDI R	QSYS	1998/12/08	15: 52: 40	*OWNER	40960	*SYSTEM
3008	QPQPDMDR	QSYS	1998/12/09	03: 14: 09	*OWNER	356352	*SYSTEM
3069	QDBCHEOD	QSYS	1998/12/22	13: 39: 30	*USER	28672	*SYSTEM
3107	QCJQHATM	QSYS	1998/12/08	07: 09: 46	*USER	24576	*SYSTEM
3165	QRFPTHRU	QSYS	1998/12/08	12: 41: 22	*OWNER	65536	*SYSTEM
3193	QOEB13	QSYS	1998/11/17	21: 34: 16	*USER	8192	*SYSTEM
3194	QOEB14	QSYS	1998/11/17	19: 24: 34	*USER	8192	*SYSTEM
3195	QOEB15	QSYS	1998/11/17	22: 57: 49	*USER	8192	*SYSTEM
3196	QOEB16	QSYS	1998/11/17	22: 58: 13	*USER	8192	*SYSTEM
3197	QLPI VPD	QSYS	1998/12/08	11: 53: 04	*USER	28672	*SYSTEM
3206	QEZPRDEV	QSYS	1998/12/08	10: 07: 17	*USER	20480	*SYSTEM
3209	QEZSAVI N	QSYS	1998/12/08	10: 09: 26	*USER	32768	*SYSTEM
3210	QMHORDQD	QSYS	1999/04/28	12: 09: 11	*USER	32768	*SYSTEM
3241	QEZBCKUP	QSYS	1998/12/08	09: 58: 46	*USER	73728	*SYSTEM
3247	QEZI NIT	QSYS	1998/12/08	10: 05: 12	*USER	40960	*SYSTEM
3249	QSRSAVO	QSYS	1999/01/13	11: 43: 32	*USER	106496	*SYSTEM
3298	QHCORYLN	QSYS	1998/12/08	09: 10: 16	*USER	24576	*SYSTEM
3315	QSPEXTWI	QSYS	1998/12/08	15: 34: 55	*OWNER	32768	*SYSTEM
3336	QRFXTL	QSYS	1998/12/08	12: 41: 12	*USER	49152	*SYSTEM
3405	QQQGET	QSYS	2000/02/11	14: 06: 48	*USER	258048	*SYSTEM
3432	QREXX	QSYS	1998/12/08	14: 55: 30	*USER	20480	*SYSTEM
3469	QLPI NATO	QSYS	1999/05/18	19: 37: 34	*OWNER	262144	*SYSTEM
3470	QLPWRKI P	QSYS	1998/12/08	08: 02: 44	*USER	40960	*SYSTEM
3471	QLPINLPP	QSYS	1999/05/18	19: 41: 56	*OWNER	393216	*SYSTEM
3474	QLPCRTDT	QSYS	1998/12/08	10: 01: 25	*USER	45056	*SYSTEM
3523	QSRMTR2	QSYS	1998/12/08	13: 46: 44	*USER	24576	*SYSTEM
3548	QPZSYNC	QSYS	1998/12/08	12: 27: 47	*OWNER	77824	*SYSTEM
3622	QPZDLOBJ	QSYS	1998/12/08	09: 34: 22	*OWNER	61440	*SYSTEM
3633	QEZCLINT	QSYS	1998/12/08	10: 03: 38	*OWNER	36864	*SYSTEM
3671	QHCCCFG	QSYS	1998/12/08	10: 50: 51	*USER	114688	*SYSTEM
3700	QEZCHGOP	QSYS	1998/12/08	07: 12: 51	*USER	40960	*SYSTEM
3701	QEZCHGPW	QSYS	1998/12/08	10: 00: 33	*USER	28672	*SYSTEM
3741	QZSPMJOB	QSYS	1998/12/08	15: 57: 04	*OWNER	40960	*SYSTEM
3745	QOHFI XQ	QSYS	1998/12/08	13: 24: 33	*OWNER	24576	*SYSTEM
3748	QEXCLRCI	QSYS	1998/12/08	09: 50: 10	*USER	16384	*SYSTEM
3758	QREXQ	QSYS	1998/12/08	14: 55: 11	*OWNER	20480	*SYSTEM
3759	QREXVAR	QSYS	1998/12/08	14: 55: 20	*USER	20480	*SYSTEM
3762	QWSSYSRQ	QSYS	1998/12/08	17: 31: 17	*USER	16384	*SYSTEM
3769	QAESTUB3	QSYS	1998/12/08	06: 20: 41	*USER	16384	*SYSTEM
3879	QLPRMPGM	QSYS	1999/01/29	23: 23: 23	*USER	49152	*USER
3880	QSRTGTRS	QSYS	1998/12/08	10: 22: 04	*OWNER	24576	*SYSTEM
3896	QECCVTEC	QSYS	1998/12/08	09: 28: 20	*OWNER	32768	*SYSTEM
3897	QECCVTEW	QSYS	1998/12/08	08: 04: 09	*USER	28672	*I NHERI T
3898	QECEDT	QSYS	1998/12/08	08: 05: 08	*USER	28672	*I NHERI T
3915	QEZBKSCD	QSYS	1998/12/08	10: 00: 45	*USER	61440	*SYSTEM
3916	QEZBKMSG	QSYS	1998/12/08	08: 31: 17	*USER	24576	*SYSTEM
3917	QEZBKWM	QSYS	1998/12/08	08: 31: 23	*USER	20480	*SYSTEM

Nbr	Name	Li b	Date	Time	Authori ty	Size	State
3928	QSPGETF	QSYS	1998/12/08	13: 15: 20	*OWNER	53248	*SYSTEM
3929	QSPPUTF	QSYS	1998/12/08	15: 39: 44	*OWNER	28672	*SYSTEM
3936	QCCWRKCC	QSYS	1998/12/08	06: 59: 25	*USER	57344	*SYSTEM
3953	QTOAPI NG	QSYS	1998/12/08	14: 21: 14	*OWNER	32768	*SYSTEM
3954	QTOASTT	QSYS	1998/12/09	03: 12: 28	*OWNER	24576	*SYSTEM
3955	QTOAUSTT	QSYS	1998/12/09	04: 31: 54	*OWNER	24576	*SYSTEM
3956	QTOAOPN	QSYS	1998/12/08	14: 20: 56	*OWNER	45056	*SYSTEM
3957	QTOASND	QSYS	1998/12/09	21: 07: 35	*OWNER	24576	*SYSTEM
3964	QTOACLS	QSYS	1998/12/09	03: 19: 03	*OWNER	24576	*SYSTEM
3977	QHFOPNF	QSYS	1998/12/08	07: 34: 43	*USER	53248	*SYSTEM
3978	QHFRDSF	QSYS	1998/12/08	10: 54: 19	*USER	40960	*SYSTEM
3979	QHFWRTSF	QSYS	1998/12/08	09: 13: 18	*USER	40960	*SYSTEM
3980	QHFCGFP	QSYS	1998/12/08	07: 32: 34	*USER	36864	*SYSTEM
3981	QHFCLOF	QSYS	1998/12/08	10: 51: 27	*USER	36864	*SYSTEM
3982	QHFRVAT	QSYS	1998/12/08	10: 54: 35	*USER	57344	*SYSTEM
3983	QHFCGAT	QSYS	1998/12/08	09: 11: 56	*USER	53248	*SYSTEM
3984	QHFDLTFS	QSYS	1998/12/08	10: 51: 17	*USER	45056	*SYSTEM
4014	QOKCCTL	QSYS	1998/12/08	13: 32: 24	*OWNER	61440	*SYSTEM
4020	QOKCCTOR	QSYS	1998/12/08	11: 15: 41	*OWNER	192512	*SYSTEM
4023	QOKDSPDP	QSYS	1998/12/08	08: 53: 15	*OWNER	28672	*SYSTEM
4025	QWPAPUT	QSYS	1999/09/01	13: 15: 08	*USER	94208	*SYSTEM
4026	QWPAPTFL	QSYS	1999/09/01	13: 17: 11	*USER	217088	*SYSTEM
4027	QWPXPOT	QSYS	1998/11/18	02: 31: 49	*USER	8192	*SYSTEM
4029	QOCCTLOF	QSYS	1998/12/08	10: 46: 41	*USER	20480	*SYSTEM
4042	QLYSETS	QSYS	1998/12/08	11: 58: 59	*USER	24576	*SYSTEM
4043	QLYGETS	QSYS	1998/12/08	08: 02: 18	*USER	20480	*SYSTEM
4044	QLYWRTBI	QSYS	1998/12/08	10: 07: 05	*USER	24576	*SYSTEM
4045	QLYRDBI	QSYS	1998/12/08	11: 58: 48	*USER	24576	*SYSTEM
4096	QDOSTRTS	QSYS	1999/05/25	14: 52: 07	*USER	53248	*SYSTEM
4097	QDOENDTS	QSYS	1998/12/08	08: 02: 43	*USER	32768	*SYSTEM
4099	QDOTRND	QSYS	1998/12/08	09: 24: 31	*USER	45056	*SYSTEM
4100	QSFTRPB	QSYS	1998/12/08	10: 31: 08	*USER	32768	*SYSTEM
4149	QPDLOGER	QSYS	1998/12/08	14: 16: 28	*USER	40960	*SYSTEM
4169	QEZSVI BM	QSYS	1998/12/08	08: 41: 14	*USER	28672	*SYSTEM
4171	QLGRTVSS	QSYS	1999/03/03	18: 38: 47	*USER	53248	*SYSTEM
4173	QLGVLI D	QSYS	1998/12/08	07: 53: 03	*USER	28672	*SYSTEM
4175	QLGCNVSS	QSYS	1998/12/08	07: 51: 49	*USER	32768	*SYSTEM
4192	QSPRTPB	QSYS	1998/12/08	10: 30: 34	*USER	20480	*SYSTEM
4207	QEZWRDSK	QSYS	1998/12/08	07: 18: 10	*USER	57344	*SYSTEM
4237	QSPFI XUP	QSYS	1998/12/18	16: 18: 53	*OWNER	81920	*SYSTEM
4252	QEZSNDMG	QSYS	1998/12/08	08: 40: 10	*USER	40960	*SYSTEM
4259	QPDWRKPB	QSYS	1998/12/08	14: 19: 07	*USER	24576	*SYSTEM
4261	QMHMRVM	QSYS	1998/12/08	12: 27: 54	*USER	65536	*SYSTEM
4262	QMHMRVPM	QSYS	1999/08/16	15: 08: 40	*USER	77824	*SYSTEM
4263	QMHRCVM	QSYS	1999/06/14	16: 59: 14	*USER	143360	*SYSTEM
4264	QMHRCVPM	QSYS	1999/08/16	15: 08: 21	*USER	184320	*SYSTEM
4265	QMHMOVPM	QSYS	1999/08/16	14: 54: 21	*USER	81920	*SYSTEM
4266	QMHRSNEM	QSYS	1999/08/16	14: 54: 47	*USER	81920	*SYSTEM
4268	QMHNSDM	QSYS	1998/12/08	10: 23: 28	*USER	36864	*SYSTEM
4269	QMHNSDPM	QSYS	1999/08/16	15: 09: 17	*USER	131072	*SYSTEM
4270	QMHNSDRM	QSYS	1998/12/08	08: 16: 30	*USER	151552	*SYSTEM
4301	QDBLNGMV	QSYS	1998/12/08	06: 23: 25	*USER	28672	*I NHERI T
4323	QPMBPCS	QSYS	1998/11/18	02: 21: 05	*OWNER	8192	*SYSTEM
4332	QFVLSTNL	QSYS	1999/01/13	11: 37: 09	*USER	77824	*SYSTEM
4345	QEXCHJCB	QSYS	1998/12/08	08: 18: 42	*USER	20480	*SYSTEM
4397	QPMASERV	QSYS	1998/12/08	14: 19: 42	*OWNER	86016	*SYSTEM
4398	QPMACLCT	QSYS	1998/12/09	02: 45: 01	*OWNER	147456	*SYSTEM
4417	QWVPDAGE	QSYS	1998/12/08	15: 40: 17	*USER	16384	*I NHERI T
4439	QZSPWI PR	QSYS	1998/12/08	15: 58: 26	*OWNER	32768	*SYSTEM
4440	QRZMI G4	QSYS	1998/11/18	02: 26: 50	*OWNER	8192	*SYSTEM
4442	QZSPWI PA	QSYS	1998/12/08	12: 00: 15	*OWNER	32768	*SYSTEM
4459	QLZAREQ	QSYS	1999/01/27	17: 57: 20	*OWNER	90112	*SYSTEM
4460	QLZARLS	QSYS	1998/12/08	10: 09: 21	*OWNER	53248	*SYSTEM
4461	QLZARTV	QSYS	1998/12/08	12: 03: 28	*USER	36864	*SYSTEM
4463	QLEDAGE	QSYS	1998/12/14	03: 25: 08	*USER	118784	*SYSTEM
4514	QTOAEND	QSYS	1998/12/09	03: 19: 11	*OWNER	20480	*SYSTEM
4515	QTOAGTI D	QSYS	1998/12/08	10: 51: 54	*OWNER	24576	*SYSTEM
4516	QTOASTRM	QSYS	1998/12/08	14: 21: 14	*OWNER	40960	*SYSTEM
4517	QTOATI ME	QSYS	1998/12/08	10: 52: 24	*OWNER	28672	*SYSTEM
4518	QTOAUOPN	QSYS	1998/12/08	16: 41: 47	*OWNER	28672	*SYSTEM
4519	QTOAUSND	QSYS	1998/12/09	04: 31: 54	*OWNER	24576	*SYSTEM
4520	QTOAUCLS	QSYS	1998/12/09	03: 19: 16	*OWNER	24576	*SYSTEM

Nbr	Name	Li b	Date	Time	Authori ty	Size	State
4521	QTOAURCV	QSYS	1998/12/09	03: 19: 46	*OWNER	24576	*SYSTEM
4522	QTOAAPI U	QSYS	1998/12/08	16: 40: 01	*OWNER	24576	*SYSTEM
4524	QTOAABRT	QSYS	1998/12/08	10: 52: 09	*OWNER	24576	*SYSTEM
4525	QTOAAPI D	QSYS	1998/12/08	16: 39: 42	*OWNER	20480	*SYSTEM
4526	QTOARCV	QSYS	1998/12/09	04: 31: 52	*OWNER	24576	*SYSTEM
4527	QTOAGHBN	QSYS	1998/12/08	14: 21: 22	*OWNER	20480	*SYSTEM
4528	QTOAI SLA	QSYS	1998/12/08	16: 41: 20	*OWNER	24576	*SYSTEM
4529	QTOAORYD	QSYS	1998/12/08	10: 51: 34	*OWNER	24576	*SYSTEM
4530	QTOAI SUP	QSYS	1998/12/08	16: 41: 11	*OWNER	24576	*SYSTEM
4540	QFVADDA	QSYS	1998/12/08	08: 56: 51	*USER	102400	*SYSTEM
4545	QFVLSTA	QSYS	1999/01/13	11: 36: 55	*USER	102400	*SYSTEM
4546	QFVRMVA	QSYS	1998/12/08	08: 57: 54	*USER	57344	*SYSTEM
4547	QFVRTVCD	QSYS	1998/12/08	07: 26: 15	*USER	69632	*SYSTEM
4552	QLI CVTTP	QSYS	1998/12/08	07: 53: 33	*USER	32768	*SYSTEM
4553	QLI COBJD	QSYS	1998/12/08	07: 56: 16	*USER	57344	*SYSTEM
4556	QEMCFG I N	QSYS	1998/12/08	09: 34: 33	*USER	24576	*SYSTEM
4578	QOTOSMAI N	QSYS	1998/12/14	02: 58: 56	*OWNER	40960	*SYSTEM
4583	QOTOSRCVR	QSYS	1998/12/14	02: 59: 01	*OWNER	45056	*SYSTEM
4636	QUSRGFI N	QSYS	1999/02/19	13: 24: 40	*USER	94208	*SYSTEM
4666	QSQLOPEN	QSYS	2000/02/24	11: 53: 42	*USER	102400	*SYSTEM
4667	QSQCLSE	QSYS	2000/02/09	12: 58: 40	*USER	49152	*SYSTEM
4668	QSQLCMI T	QSYS	1999/07/16	09: 37: 43	*USER	32768	*SYSTEM
4746	QPZPTFI N	QSYS	1998/12/09	03: 12: 31	*OWNER	77824	*SYSTEM
4791	QFPAACTV	QSYS	1998/12/14	02: 34: 49	*OWNER	40960	*SYSTEM
4792	QFPAMON	QSYS	2000/01/17	08: 22: 33	*OWNER	98304	*SYSTEM
4829	QESDPSA	QSYS	1998/12/08	08: 14: 48	*OWNER	24576	*SYSTEM
4832	QESRSRVA	QSYS	1998/12/08	09: 43: 50	*OWNER	36864	*SYSTEM
4833	QESCSRVA	QSYS	1998/12/08	09: 41: 28	*USER	36864	*SYSTEM
4834	QESSAVMA	QSYS	1998/12/08	09: 42: 45	*USER	20480	*SYSTEM
4835	QMHCHGEM	QSYS	1999/08/16	14: 54: 52	*USER	36864	*SYSTEM
4836	QMHJJOBL	QSYS	1999/01/13	11: 33: 38	*OWNER	155648	*SYSTEM
4837	QMHLSM	QSYS	1999/01/13	11: 34: 11	*USER	159744	*SYSTEM
4838	QMHPRMM	QSYS	1998/12/08	10: 20: 58	*USER	45056	*SYSTEM
4839	QMHRTVM	QSYS	1998/12/08	08: 14: 16	*USER	40960	*SYSTEM
4840	QMHMQAT	QSYS	1998/12/08	08: 15: 13	*USER	40960	*SYSTEM
4841	QMHSNDBM	QSYS	1998/12/08	12: 27: 49	*USER	32768	*SYSTEM
4842	QMHMFAT	QSYS	1998/12/08	10: 22: 39	*USER	32768	*SYSTEM
4843	QUHDSPH	QSYS	1998/12/08	14: 49: 58	*USER	32768	*SYSTEM
4844	QUHPRTH	QSYS	1998/12/08	17: 03: 41	*USER	24576	*SYSTEM
4845	QUI ADDLE	QSYS	1998/12/08	17: 03: 47	*USER	28672	*SYSTEM
4846	QUI ADDLM	QSYS	1998/12/08	14: 50: 55	*USER	28672	*SYSTEM
4847	QUI ADDPA	QSYS	1998/12/08	14: 51: 31	*USER	28672	*SYSTEM
4848	QUI ADDPW	QSYS	1998/12/08	11: 14: 52	*USER	28672	*SYSTEM
4849	QUI CLOA	QSYS	1998/12/08	14: 52: 12	*USER	28672	*SYSTEM
4850	QUI DLT L	QSYS	1998/12/08	17: 05: 07	*USER	24576	*SYSTEM
4851	QUI DSPP	QSYS	1998/12/08	14: 52: 15	*USER	28672	*SYSTEM
4852	QUI GETLE	QSYS	1998/12/08	17: 05: 31	*USER	28672	*SYSTEM
4853	QUI GETLM	QSYS	1998/12/08	14: 53: 17	*USER	28672	*SYSTEM
4854	QUI GETV	QSYS	1998/12/08	14: 53: 11	*USER	24576	*SYSTEM
4855	QUI OPNDA	QSYS	1998/12/08	11: 16: 42	*USER	28672	*SYSTEM
4856	QUI OPNPA	QSYS	1998/12/08	17: 06: 32	*USER	28672	*SYSTEM
4857	QUI PRTP	QSYS	1998/12/08	11: 17: 51	*USER	24576	*SYSTEM
4858	QUI RMVLE	QSYS	1998/12/08	14: 56: 19	*USER	24576	*SYSTEM
4859	QUI RMVPA	QSYS	1998/12/08	14: 56: 06	*USER	28672	*SYSTEM
4860	QUI RMVPW	QSYS	1998/12/08	11: 18: 32	*USER	24576	*SYSTEM
4861	QUI SETLA	QSYS	1998/12/08	14: 57: 23	*USER	24576	*SYSTEM
4862	QUI SETSC	QSYS	1998/12/08	11: 17: 52	*USER	24576	*SYSTEM
4863	QUI UPDL E	QSYS	1998/12/08	17: 08: 41	*USER	28672	*SYSTEM
4865	QSYCHGPR	QSYS	1998/12/08	16: 17: 55	*OWNER	28672	*SYSTEM
4866	QSYCHGPW	QSYS	1999/08/25	10: 12: 28	*OWNER	61440	*SYSTEM
4867	QSYCUSRA	QSYS	1998/12/08	10: 32: 18	*OWNER	57344	*SYSTEM
4868	QSYCVTA	QSYS	1998/12/08	16: 18: 04	*USER	28672	*SYSTEM
4869	QSYGETPH	QSYS	1999/01/13	13: 43: 39	*OWNER	40960	*SYSTEM
4870	QSYLAUTU	QSYS	1999/01/13	11: 43: 58	*USER	57344	*SYSTEM
4871	QSYLATLO	QSYS	1999/01/13	11: 43: 49	*OWNER	86016	*SYSTEM
4872	QSYLOBJP	QSYS	1999/01/13	11: 44: 20	*OWNER	61440	*SYSTEM
4873	QSYLOBJA	QSYS	1999/01/13	11: 44: 10	*USER	77824	*SYSTEM
4874	QSYLUSRA	QSYS	1999/01/13	11: 45: 19	*OWNER	77824	*SYSTEM
4875	QSYRLSPH	QSYS	1998/12/08	13: 53: 47	*OWNER	28672	*SYSTEM
4876	QSYRUSRA	QSYS	1998/12/08	10: 36: 46	*OWNER	57344	*SYSTEM
4877	QSYRUSRI	QSYS	1998/12/09	04: 29: 42	*USER	53248	*SYSTEM
4878	QMHREQM	QSYS	1999/04/28	12: 09: 20	*USER	45056	*SYSTEM

Nbr	Name	Li b	Date	Time	Authori ty	Size	State
4879	QJOSJRNE	QSYS	1998/12/08	07: 50: 41	*USER	36864	*SYSTEM
4880	QTNRCMTI	QSYS	1998/12/08	14: 17: 56	*USER	32768	*SYSTEM
4881	QTNADDCR	QSYS	1998/12/08	10: 48: 01	*USER	57344	*SYSTEM
4882	QTNRMVCR	QSYS	1998/12/08	16: 38: 43	*USER	32768	*SYSTEM
4883	QDMRTVFO	QSYS	1999/04/02	12: 36: 28	*USER	32768	*SYSTEM
4885	QOKSCHD	QSYS	1998/12/08	08: 57: 45	*OWNER	229376	*SYSTEM
4888	QFPHDLSS	QSYS	1998/12/08	07: 23: 52	*OWNER	24576	*SYSTEM
4898	QUSRJOBI	QSYS	1998/12/09	03: 31: 27	*OWNER	81920	*SYSTEM
4905	QZPAI JOB	QSYS	1998/12/08	15: 54: 03	*OWNER	53248	*SYSTEM
4914	QUSRTVEI	QSYS	1998/12/14	03: 05: 44	*USER	36864	*I NHERI T
4915	QUSRGPT	QSYS	1998/12/14	03: 05: 33	*USER	36864	*I NHERI T
4916	QUSDRGPT	QSYS	1998/12/14	03: 05: 18	*USER	36864	*I NHERI T
4917	QUSADDEP	QSYS	1998/12/14	03: 05: 13	*USER	36864	*I NHERI T
4918	QUSRMVEP	QSYS	1998/12/14	03: 05: 38	*USER	36864	*I NHERI T
4926	QDCRNWSD	QSYS	1998/12/08	08: 46: 46	*USER	69632	*SYSTEM
4928	QUSCHGPA	QSYS	1998/12/08	14: 58: 59	*USER	40960	*SYSTEM
4929	QUSLJOB	QSYS	1999/01/13	11: 42: 32	*OWNER	147456	*SYSTEM
4931	QWCCCJOB	QSYS	1998/12/08	15: 08: 22	*USER	32768	*SYSTEM
4932	QWCCHGTN	QSYS	1998/12/08	15: 10: 09	*USER	32768	*SYSTEM
4933	QWCCVTDI	QSYS	1998/12/08	15: 11: 56	*USER	40960	*SYSTEM
4934	QWCLASBS	QSYS	1999/01/13	11: 41: 03	*OWNER	53248	*SYSTEM
4935	QWCLOBJL	QSYS	1999/01/13	11: 41: 14	*OWNER	73728	*SYSTEM
4936	QWCLSCDE	QSYS	1999/01/13	11: 41: 27	*OWNER	81920	*SYSTEM
4937	QWCRDTAA	QSYS	1998/12/08	17: 21: 32	*USER	32768	*SYSTEM
4938	QWCRNETA	QSYS	1998/12/08	11: 30: 16	*USER	45056	*SYSTEM
4939	QWCRSSTS	QSYS	1998/12/08	17: 22: 32	*USER	73728	*SYSTEM
4940	QWCRSVAL	QSYS	1998/12/08	17: 21: 51	*USER	73728	*SYSTEM
4941	QWDL SJBQ	QSYS	1999/01/13	11: 41: 48	*OWNER	65536	*SYSTEM
4942	QWDRJOB	QSYS	1998/12/08	11: 33: 29	*USER	36864	*SYSTEM
4943	QWDRSBS	QSYS	1998/12/08	17: 25: 30	*OWNER	36864	*SYSTEM
4944	QWTSETP	QSYS	1998/12/08	11: 48: 41	*OWNER	40960	*SYSTEM
4946	QLI CHGLL	QSYS	1998/12/08	09: 53: 26	*USER	32768	*SYSTEM
4947	QLI RLI BD	QSYS	1998/12/08	07: 55: 42	*USER	40960	*SYSTEM
4948	QLI RNMO	QSYS	1998/12/09	02: 44: 51	*USER	65536	*SYSTEM
4950	QUSLOBJ	QSYS	1999/01/13	11: 39: 43	*OWNER	81920	*SYSTEM
4951	QUSROBJD	QSYS	1998/12/08	11: 20: 40	*USER	36864	*SYSTEM
4954	QZSPWLOC	QSYS	1998/12/08	17: 55: 03	*OWNER	32768	*SYSTEM
4958	QDCPOLP	QSYS	1998/12/08	06: 41: 22	*USER	20480	*USER
4981	QLGTRDTA	QSYS	1998/12/08	09: 52: 13	*USER	36864	*SYSTEM
5030	QZDAGFS	QSYS	1998/11/18	04: 20: 44	*USER	8192	*SYSTEM
5040	QPDAUTP2	QSYS	1998/12/08	14: 16: 28	*OWNER	20480	*SYSTEM
5044	QPDGENSS	QSYS	1998/12/14	02: 30: 40	*USER	73728	*SYSTEM
5048	QUSRSPLA	QSYS	1998/12/08	17: 11: 28	*USER	65536	*SYSTEM
5049	QUSLSPL	QSYS	1999/05/04	17: 01: 38	*OWNER	110592	*SYSTEM
5050	QSPROUTQ	QSYS	1999/07/15	12: 07: 57	*OWNER	49152	*SYSTEM
5051	QSPRJOBQ	QSYS	1998/12/08	15: 41: 02	*OWNER	40960	*SYSTEM
5052	QSPOPNSP	QSYS	1998/12/18	16: 19: 12	*USER	40960	*SYSTEM
5053	QSPCLOSP	QSYS	1998/12/18	16: 16: 57	*OWNER	32768	*SYSTEM
5054	QSPCRTSP	QSYS	1999/05/04	17: 02: 51	*OWNER	225280	*SYSTEM
5055	QSPGETSP	QSYS	1999/01/13	11: 34: 37	*OWNER	81920	*SYSTEM
5056	QSPPUTSP	QSYS	1999/01/13	11: 42: 58	*OWNER	81920	*SYSTEM
5057	QSPMOVSP	QSYS	1998/12/08	15: 36: 58	*USER	36864	*SYSTEM
5058	QSPMOVJB	QSYS	1998/12/08	10: 09: 22	*OWNER	69632	*SYSTEM
5059	QZCAADDC	QSYS	1998/12/14	03: 08: 21	*OWNER	36864	*USER
5060	QZCARMVC	QSYS	1998/12/14	03: 08: 36	*OWNER	36864	*USER
5061	QZCAREFC	QSYS	1998/12/14	03: 08: 31	*OWNER	36864	*USER
5062	QZCAUPDC	QSYS	1998/12/14	03: 08: 46	*OWNER	36864	*USER
5063	QLGCNVCS	QSYS	1998/12/14	02: 33: 00	*USER	36864	*I NHERI T
5064	QSQCHKS	QSYS	1999/06/19	15: 16: 37	*USER	69632	*SYSTEM
5065	QZMFACHG	QSYS	1998/12/08	17: 49: 41	*OWNER	204800	*SYSTEM
5066	QZMFACRT	QSYS	1998/12/08	15: 51: 00	*OWNER	135168	*SYSTEM
5067	QZMFADDC	QSYS	1998/12/08	15: 50: 34	*OWNER	40960	*SYSTEM
5069	QZMFALOG	QSYS	1998/11/18	04: 25: 40	*USER	8192	*SYSTEM
5070	QZMFAPG1	QSYS	1998/11/18	02: 43: 13	*USER	8192	*SYSTEM
5071	QZMFARSV	QSYS	1998/12/08	17: 49: 17	*OWNER	32768	*SYSTEM
5072	QZMFARTV	QSYS	1998/12/08	15: 51: 29	*OWNER	49152	*SYSTEM
5073	QZMFASCR	QSYS	1998/12/08	15: 51: 06	*OWNER	32768	*SYSTEM
5074	QZMFASQC	QSYS	1998/12/08	11: 55: 31	*OWNER	32768	*SYSTEM
5075	QZMFCATR	QSYS	1998/11/18	04: 26: 13	*USER	8192	*SYSTEM
5076	QZMFDLTC	QSYS	1998/12/08	17: 50: 15	*OWNER	40960	*SYSTEM
5077	QZMFLSTC	QSYS	1999/01/13	11: 32: 44	*OWNER	65536	*SYSTEM
5078	QSYCURS	QSYS	1998/12/08	16: 18: 36	*OWNER	36864	*SYSTEM

Nbr	Name	Li b	Date	Time	Authori ty	Size	State
5079	QALGENA	QSYS	1998/12/08	05: 42: 13	*USER	32768	*SYSTEM
5080	QALRTVA	QSYS	1998/12/08	06: 30: 19	*USER	36864	*SYSTEM
5081	QALSND	QSYS	1998/12/08	05: 43: 34	*USER	40960	*SYSTEM
5082	QBNLPGMI	QSYS	1999/01/13	11: 35: 30	*OWNER	122880	*SYSTEM
5083	QBNLSPGM	QSYS	1999/01/13	11: 35: 53	*OWNER	131072	*SYSTEM
5084	QBNRSPGM	QSYS	1998/12/08	05: 56: 18	*USER	49152	*SYSTEM
5085	QCAPCMD	QSYS	1998/12/08	06: 02: 23	*USER	49152	*SYSTEM
5086	QCDRCMDI	QSYS	1998/12/08	06: 56: 55	*USER	32768	*SYSTEM
5087	QCLRPGAS	QSYS	1998/12/08	06: 13: 21	*USER	36864	*SYSTEM
5088	QCLRPGMI	QSYS	1998/12/09	01: 52: 35	*USER	69632	*SYSTEM
5089	QCLSPGAS	QSYS	1998/12/08	06: 14: 23	*USER	40960	*SYSTEM
5092	QDBRTVFD	QSYS	1999/05/21	10: 46: 22	*USER	81920	*SYSTEM
5093	QUSLMBR	QSYS	1999/03/23	22: 00: 24	*OWNER	106496	*SYSTEM
5094	QUSRMBRD	QSYS	1999/02/04	14: 25: 31	*USER	98304	*SYSTEM
5095	QDCLCFGD	QSYS	1999/01/13	11: 36: 21	*USER	61440	*SYSTEM
5096	QDCRCFGS	QSYS	1998/12/08	07: 30: 13	*USER	49152	*SYSTEM
5097	QDCRCTLD	QSYS	1998/12/08	07: 32: 37	*USER	192512	*SYSTEM
5098	QDCRDEV	QSYS	1999/03/25	18: 04: 29	*USER	221184	*SYSTEM
5099	QDCRLI ND	QSYS	1999/03/30	18: 07: 53	*USER	438272	*SYSTEM
5100	QDFRTVFD	QSYS	1998/12/08	07: 49: 58	*USER	40960	*SYSTEM
5101	QEARMVBM	QSYS	1998/12/08	09: 26: 50	*USER	32768	*SYSTEM
5102	QEZLSGNU	QSYS	1999/01/13	11: 36: 34	*USER	77824	*SYSTEM
5103	QHFCLODR	QSYS	1998/12/08	10: 51: 26	*USER	36864	*SYSTEM
5104	QHFCPYSF	QSYS	1998/12/08	09: 11: 11	*USER	61440	*SYSTEM
5105	QHFCRTDR	QSYS	1998/12/08	09: 11: 40	*USER	53248	*SYSTEM
5106	QHFCFLFS	QSYS	1998/12/08	07: 34: 04	*USER	53248	*SYSTEM
5107	QHFFRCSF	QSYS	1998/12/08	10: 52: 05	*USER	36864	*SYSTEM
5108	QHFGETSZ	QSYS	1998/12/08	09: 13: 40	*USER	36864	*SYSTEM
5109	QHFLSTFS	QSYS	1999/01/13	11: 38: 41	*USER	53248	*SYSTEM
5110	QHFLULSF	QSYS	1998/12/08	10: 51: 53	*USER	36864	*SYSTEM
5112	OZCATHR	QSYS	1998/12/14	03: 08: 41	*OWNER	102400	*SYSTEM
5148	QSOCWI FC	QSYS	1998/12/08	15: 29: 29	*OWNER	36864	*SYSTEM
5150	QSOCWRTE	QSYS	1998/12/08	13: 10: 14	*OWNER	36864	*SYSTEM
5197	QSPSNDWM	QSYS	1998/12/08	15: 42: 52	*OWNER	36864	*SYSTEM
5218	QSOCI P03	QSYS	1998/12/08	13: 08: 47	*OWNER	36864	*SYSTEM
5219	QSOCI P04	QSYS	1998/12/08	13: 07: 08	*OWNER	45056	*SYSTEM
5231	QTQRECOV	QSYS	1998/12/08	10: 59: 36	*USER	24576	*SYSTEM
5249	QZMFXP2G	QSYS	1998/11/18	02: 46: 09	*USER	16384	*SYSTEM
5250	QZMFXP3G	QSYS	1998/11/18	02: 46: 28	*USER	8192	*SYSTEM
5251	QZMFXP4G	QSYS	1998/11/17	22: 14: 58	*USER	8192	*SYSTEM
5262	QZMFXP6G	QSYS	1998/12/08	17: 50: 51	*OWNER	20480	*SYSTEM
5276	QESANOTE	QSYS	1998/12/08	08: 12: 17	*OWNER	32768	*SYSTEM
5311	QYYCCLoS	QSYS	1998/12/08	11: 50: 41	*USER	69632	*SYSTEM
5312	QYYCGETS	QSYS	1998/12/09	03: 28: 44	*USER	61440	*SYSTEM
5314	QYYCGETM	QSYS	1998/12/09	04: 49: 56	*USER	61440	*SYSTEM
5315	QYYCPUT	QSYS	1998/12/09	03: 37: 08	*USER	53248	*SYSTEM
5316	QYYCPUTM	QSYS	1998/12/09	04: 51: 01	*USER	57344	*SYSTEM
5317	QYYCPUTD	QSYS	1998/12/09	03: 37: 07	*USER	53248	*SYSTEM
5318	QYYCFEOD	QSYS	1998/12/09	03: 36: 17	*USER	57344	*SYSTEM
5319	QYYCUDR	QSYS	1998/12/09	04: 51: 01	*USER	61440	*SYSTEM
5321	QHFMVVSF	QSYS	1998/12/08	09: 12: 58	*USER	61440	*SYSTEM
5322	QHFOPNDR	QSYS	1998/12/08	09: 12: 59	*USER	53248	*SYSTEM
5323	QHFRDDR	QSYS	1998/12/08	10: 53: 26	*USER	40960	*SYSTEM
5324	QHFRGFS	QSYS	1998/12/08	09: 13: 05	*USER	49152	*SYSTEM
5325	QHFRNMDR	QSYS	1998/12/08	09: 12: 42	*USER	49152	*SYSTEM
5326	QHFRNMSF	QSYS	1998/12/08	07: 34: 33	*USER	49152	*SYSTEM
5327	QHFSSETZ	QSYS	1998/12/08	09: 15: 05	*USER	36864	*SYSTEM
5328	QLGRTVLI	QSYS	1998/12/08	09: 51: 55	*USER	32768	*SYSTEM
5329	QLGSCNMX	QSYS	1998/12/08	07: 52: 34	*USER	28672	*SYSTEM
5330	QLGSHORT	QSYS	1998/12/08	11: 37: 27	*USER	57344	*SYSTEM
5331	QLGSRTI O	QSYS	1998/12/09	02: 27: 37	*USER	143360	*SYSTEM
5332	QLPLPRDS	QSYS	1999/01/13	11: 34: 56	*USER	69632	*SYSTEM
5333	QLZAADDK	QSYS	1998/12/08	08: 03: 18	*USER	32768	*SYSTEM
5334	QLZAADDL	QSYS	1998/12/08	11: 59: 48	*OWNER	53248	*SYSTEM
5335	QLZARTVK	QSYS	1998/12/08	10: 09: 36	*USER	32768	*SYSTEM
5336	QMHSNDMS	QSYS	1999/08/16	14: 53: 59	*USER	36864	*SYSTEM
5337	QNMCHGMN	QSYS	1998/12/09	02: 27: 24	*USER	32768	*SYSTEM
5338	QNMDRGAP	QSYS	1998/12/09	02: 34: 09	*USER	32768	*SYSTEM
5339	QNMDRGFN	QSYS	1998/12/08	12: 46: 25	*OWNER	40960	*SYSTEM
5340	QBNLMODI	QSYS	1999/01/13	11: 35: 09	*USER	81920	*SYSTEM
5341	QNMDRGTI	QSYS	1998/12/09	02: 34: 11	*USER	32768	*SYSTEM
5342	QNMENDAP	QSYS	1998/12/09	02: 55: 12	*USER	32768	*SYSTEM



Nbr	Name	Li b	Date	Time	Authori ty	Size	State
5343	QNMRCVDT	QSYS	1998/12/09	02: 35: 07	*USER	45056	*SYSTEM
5344	QNMRCVOC	QSYS	1998/12/09	02: 35: 06	*USER	32768	*SYSTEM
5345	QNMREGAP	QSYS	1998/12/09	02: 55: 29	*USER	32768	*SYSTEM
5346	QNMRGFN	QSYS	1998/12/08	12: 48: 43	*OWNER	45056	*SYSTEM
5347	QNMRTGI	QSYS	1999/01/13	11: 34: 24	*USER	73728	*SYSTEM
5348	QNMRRGF	QSYS	1998/12/08	10: 40: 28	*OWNER	40960	*SYSTEM
5349	QNMRTVMN	QSYS	1998/12/09	02: 28: 22	*USER	28672	*SYSTEM
5350	QNMSENDR	QSYS	1998/12/09	02: 35: 06	*USER	32768	*SYSTEM
5351	QNMSENDRP	QSYS	1998/12/09	02: 36: 07	*USER	40960	*SYSTEM
5352	QNMSENDRQ	QSYS	1998/12/09	02: 56: 35	*USER	40960	*SYSTEM
5353	QNMSTRAP	QSYS	1998/12/09	02: 56: 58	*USER	36864	*SYSTEM
5354	QOGCHGOE	QSYS	1998/12/08	13: 25: 02	*OWNER	49152	*SYSTEM
5355	QOGRVVOE	QSYS	1998/12/08	11: 08: 29	*OWNER	36864	*SYSTEM
5356	QOKDSPX4	QSYS	1998/12/08	13: 31: 22	*OWNER	32768	*SYSTEM
5357	QPMLPFRD	QSYS	1998/12/08	14: 22: 18	*USER	36864	*SYSTEM
5358	QPMWCOL	QSYS	1999/08/17	16: 55: 34	*OWNER	53248	*SYSTEM
5359	QPRCRTPG	QSYS	1998/12/08	12: 19: 18	*USER	49152	*SYSTEM
5360	QPZCRTFX	QSYS	1998/12/08	14: 36: 47	*USER	73728	*SYSTEM
5361	QPZGENNM	QSYS	1998/12/08	14: 37: 06	*USER	36864	*SYSTEM
5362	QPZLOGFX	QSYS	1998/12/08	12: 24: 23	*USER	36864	*SYSTEM
5363	QQQORY	QSYS	1999/08/30	07: 59: 08	*USER	20480	*SYSTEM
5364	QSPRWTRI	QSYS	1998/12/08	13: 19: 59	*OWNER	28672	*SYSTEM
5365	QSQPRCED	QSYS	2000/03/13	16: 48: 02	*USER	143360	*SYSTEM
5366	QSRLSAVF	QSYS	1999/01/13	11: 43: 08	*USER	57344	*SYSTEM
5367	QSZCRTPD	QSYS	1998/12/08	16: 24: 32	*USER	61440	*SYSTEM
5368	QSZCRTPL	QSYS	1998/12/08	13: 57: 09	*USER	69632	*SYSTEM
5369	QSZDLTPD	QSYS	1998/12/08	13: 57: 36	*USER	32768	*SYSTEM
5370	QSZDLTPL	QSYS	1998/12/08	10: 37: 40	*USER	32768	*SYSTEM
5371	QSZPKGPO	QSYS	1998/12/08	13: 57: 59	*USER	32768	*SYSTEM
5372	QSZRTVPR	QSYS	1998/12/08	10: 38: 24	*USER	73728	*SYSTEM
5373	QSZSLTPR	QSYS	1998/12/08	16: 25: 28	*OWNER	65536	*SYSTEM
5374	QTERTVPV	QSYS	1998/12/08	14: 12: 28	*USER	49152	*SYSTEM
5375	QTNCHGCO	QSYS	1998/12/08	16: 37: 05	*USER	36864	*SYSTEM
5376	QTNBRBOD	QSYS	1998/12/08	16: 38: 56	*USER	36864	*SYSTEM
5377	QTVCLQVT	QSYS	1998/12/09	03: 26: 03	*USER	28672	*SYSTEM
5378	QTVOPNVT	QSYS	1998/12/09	04: 35: 00	*USER	53248	*SYSTEM
5379	QTVRQVT	QSYS	1998/12/09	03: 19: 07	*USER	32768	*SYSTEM
5380	QTVSNDRO	QSYS	1998/12/09	04: 35: 35	*USER	32768	*SYSTEM
5381	QTVWRTVT	QSYS	1998/12/09	03: 29: 48	*USER	36864	*SYSTEM
5382	QTWAI DSP	QSYS	1999/04/21	14: 15: 41	*OWNER	49152	*SYSTEM
5383	QBNRMODI	QSYS	1998/12/09	01: 47: 10	*USER	69632	*SYSTEM
5384	QTWCHKSP	QSYS	1999/04/21	14: 15: 32	*OWNER	49152	*SYSTEM
5385	QUI PUTV	QSYS	1998/12/08	17: 07: 06	*USER	24576	*SYSTEM
5386	QUI RTVLA	QSYS	1998/12/08	17: 08: 01	*USER	24576	*SYSTEM
5387	QUSADDUI	QSYS	1999/01/13	11: 44: 46	*USER	49152	*SYSTEM
5388	QUSCHGUS	QSYS	1999/01/13	11: 44: 39	*USER	53248	*SYSTEM
5389	QUSCRTUI	QSYS	1998/12/08	17: 10: 33	*USER	53248	*SYSTEM
5390	QUSCRTUQ	QSYS	1998/12/08	14: 59: 29	*USER	53248	*SYSTEM
5391	QUSCRTUS	QSYS	1999/03/24	16: 12: 15	*USER	53248	*SYSTEM
5392	QUSCUSAT	QSYS	1999/01/13	11: 44: 31	*USER	53248	*SYSTEM
5393	QUSDLTUI	QSYS	1998/12/08	17: 10: 20	*USER	28672	*SYSTEM
5394	QUSDLTUQ	QSYS	1998/12/08	17: 10: 11	*USER	28672	*SYSTEM
5395	QUSDLTUS	QSYS	1998/12/08	14: 59: 20	*USER	28672	*SYSTEM
5396	QUSPTRUS	QSYS	1998/12/08	17: 10: 39	*USER	36864	*SYSTEM
5397	QUSRMVUI	QSYS	1999/09/29	12: 09: 37	*USER	57344	*SYSTEM
5398	QUSRTVUI	QSYS	1999/09/29	12: 09: 27	*USER	57344	*SYSTEM
5399	QUSRTVUS	QSYS	1999/01/13	11: 40: 41	*USER	53248	*SYSTEM
5400	QUSRUI AT	QSYS	1999/01/13	11: 40: 47	*USER	49152	*SYSTEM
5401	QUSRUSAT	QSYS	1999/03/22	09: 12: 38	*USER	53248	*SYSTEM
5402	QVTRMSTG	QSYS	1998/12/08	17: 14: 43	*OWNER	28672	*SYSTEM
5403	QDBLDBR	QSYS	1999/01/13	11: 36: 12	*OWNER	73728	*SYSTEM
5404	QUSLFLD	QSYS	1999/01/30	11: 39: 55	*OWNER	77824	*SYSTEM
5405	QUSLRCD	QSYS	1999/01/13	11: 39: 54	*OWNER	73728	*SYSTEM
5406	QWPZTAFF	QSYS	1998/12/08	17: 28: 31	*USER	20480	*SYSTEM
5407	QWTCTLTR	QSYS	1998/12/08	17: 34: 01	*USER	28672	*SYSTEM
5408	QWTDMPFR	QSYS	1998/12/08	17: 34: 28	*USER	28672	*SYSTEM
5409	QWTDMPLF	QSYS	1998/12/09	03: 28: 37	*USER	65536	*SYSTEM
5410	QWTSETLF	QSYS	1998/12/09	04: 46: 35	*USER	28672	*SYSTEM
5411	QWTSETTR	QSYS	1998/12/08	17: 40: 53	*USER	28672	*SYSTEM
5412	QEZAST	QSYS	1998/12/12	04: 26: 22	*USER	28672	*USER
5413	QEZBCHJB	QSYS	1998/12/12	04: 26: 23	*USER	28672	*USER
5414	QEZMSG	QSYS	1998/12/12	04: 26: 34	*USER	28672	*USER

Nbr	Name	Li b	Date	Time	Authori ty	Size	State
5415	QEZOUTPT	QSYS	1998/12/12	04: 26: 34	*USER	28672	*USER
5416	QLRCHGCM	QSYS	1998/12/09	20: 16: 36	*USER	28672	*I NHERI T
5417	QLRRTVCE	QSYS	1998/12/09	20: 17: 00	*USER	28672	*I NHERI T
5418	QLRSETCE	QSYS	1998/12/09	20: 13: 52	*USER	36864	*I NHERI T
5419	QLZAGENK	QSYS	1998/12/08	10: 08: 04	*OWNER	49152	*SYSTEM
5420	QMHCTLJL	QSYS	1998/12/08	10: 17: 01	*USER	49152	*SYSTEM
5421	QMARQSOA	QSYS	1998/12/08	10: 14: 20	*OWNER	36864	*SYSTEM
5422	QHFDRGFS	QSYS	1998/12/08	09: 11: 58	*USER	40960	*SYSTEM
5423	QJORJIDI	QSYS	1998/12/08	09: 45: 25	*USER	49152	*SYSTEM
5424	QTQCVRT	QSYS	1999/01/15	17: 32: 10	*OWNER	49152	*SYSTEM
5425	QTQGCCN	QSYS	1998/12/08	16: 50: 21	*OWNER	24576	*SYSTEM
5426	QTQGESP	QSYS	1998/12/08	14: 33: 09	*OWNER	24576	*SYSTEM
5427	QTQGRDC	QSYS	1998/12/08	16: 49: 58	*OWNER	24576	*SYSTEM
5428	QTQSCSP	QSYS	1998/12/08	16: 50: 47	*OWNER	24576	*SYSTEM
5429	QPZRTVFX	QSYS	1998/12/08	09: 35: 43	*OWNER	77824	*SYSTEM
5430	QZMFAQRY	QSYS	1998/12/08	17: 48: 49	*OWNER	32768	*SYSTEM
5431	QHFDLTDR	QSYS	1998/12/08	10: 51: 50	*USER	45056	*SYSTEM
5432	QWPZHPTR	QSYS	1998/12/14	03: 05: 49	*USER	36864	*I NHERI T
5433	QSPCHGOO	QSYS	1999/07/01	16: 12: 25	*USER	32768	*SYSTEM
5434	QSPBSEPP	QSYS	1998/12/08	15: 32: 02	*USER	53248	*SYSTEM
5435	QSPBOPNC	QSYS	1998/12/08	13: 08: 18	*USER	32768	*SYSTEM
5436	QDCCCFGD	QSYS	1998/12/08	06: 56: 35	*USER	45056	*SYSTEM
5437	QGLSLRSC	QSYS	1999/01/13	11: 38: 31	*OWNER	200704	*SYSTEM
5438	QGSCPYRS	QSYS	1999/01/13	11: 37: 18	*USER	65536	*SYSTEM
5439	QLPCDINF	QSYS	1998/12/14	03: 02: 52	*USER	36864	*I NHERI T
5440	QLPCDRST	QSYS	1998/12/14	03: 02: 57	*USER	36864	*I NHERI T
5441	QLGRTVLC	QSYS	1998/12/14	03: 02: 41	*USER	40960	*I NHERI T
5442	QRZCRTH	QSYS	1998/12/08	09: 51: 47	*USER	32768	*SYSTEM
5443	QTQSMXC	QSYS	1998/12/08	14: 34: 11	*USER	20480	*I NHERI T
5444	QTQGCTL	QSYS	1998/12/08	14: 32: 58	*OWNER	45056	*SYSTEM
5445	QRZDLTE	QSYS	1998/12/08	12: 48: 50	*USER	28672	*SYSTEM
5446	QRZCHGE	QSYS	1998/12/08	15: 04: 27	*USER	28672	*SYSTEM
5447	QRZDLTH	QSYS	1998/12/08	12: 49: 01	*USER	28672	*SYSTEM
5448	QRZRRCR	QSYS	1998/12/08	15: 11: 07	*USER	159744	*SYSTEM
5449	QZDCRFSO	QSYS	1998/12/08	17: 45: 16	*OWNER	36864	*SYSTEM
5450	QZDWTFSO	QSYS	1998/12/08	17: 48: 17	*OWNER	40960	*SYSTEM
5451	QZDRDFSO	QSYS	1998/12/08	15: 48: 08	*OWNER	40960	*SYSTEM
5452	QZDASNI D	QSYS	1998/12/08	11: 51: 04	*OWNER	32768	*SYSTEM
5453	QZDRVKI D	QSYS	1998/12/08	17: 46: 58	*OWNER	32768	*SYSTEM
5454	QZDLSTI D	QSYS	1999/01/13	11: 41: 57	*OWNER	61440	*SYSTEM
5455	QZDRTVI D	QSYS	1998/12/08	15: 48: 31	*OWNER	36864	*SYSTEM
5456	QLPI SLNG	QSYS	1998/12/08	08: 01: 02	*USER	40960	*SYSTEM
5457	QWSRTVOI	QSYS	2000/02/23	11: 55: 45	*USER	53248	*SYSTEM
5458	QRZRRSI	QSYS	1998/12/08	12: 52: 02	*USER	98304	*SYSTEM
5459	QRZRTVR	QSYS	1998/12/08	15: 11: 19	*USER	32768	*SYSTEM
5460	QRZSCHE	QSYS	1998/12/08	12: 52: 50	*USER	118784	*SYSTEM
5594	QWTATNAD	QSYS	1998/12/08	15: 30: 02	*OWNER	20480	*SYSTEM
5661	QFPAMONB	QSYS	1998/12/14	02: 35: 11	*OWNER	102400	*SYSTEM
5662	QFPAMONN	QSYS	1998/12/14	02: 35: 16	*OWNER	114688	*SYSTEM
5671	QOKADDDP	QSYS	1998/12/08	11: 12: 55	*OWNER	28672	*SYSTEM
5672	QOKCHGDP	QSYS	1998/12/08	13: 29: 35	*OWNER	28672	*SYSTEM
5673	QOKRMVDP	QSYS	1998/12/08	13: 35: 21	*OWNER	28672	*SYSTEM
5674	QUI LNGTX	QSYS	1998/12/08	11: 15: 52	*USER	32768	*SYSTEM
5675	QTALCTG	QSYS	1998/11/18	02: 52: 54	*USER	8192	*SYSTEM
5676	QTAPI PE	QSYS	1998/11/18	02: 53: 09	*USER	8192	*SYSTEM
5677	QTARDCAP	QSYS	1998/12/08	16: 31: 08	*USER	61440	*SYSTEM
5678	QDBRTVSN	QSYS	1999/05/27	18: 42: 04	*OWNER	57344	*SYSTEM
5679	QEZCHBKL	QSYS	1998/12/08	08: 32: 33	*USER	40960	*SYSTEM
5680	QEZCHBKS	QSYS	1999/05/25	14: 48: 59	*USER	73728	*SYSTEM
5681	QEZRTBKD	QSYS	1999/05/25	14: 48: 24	*USER	45056	*SYSTEM
5682	QEZRTBKH	QSYS	1999/05/25	14: 48: 29	*USER	36864	*SYSTEM
5683	QEZRTBKO	QSYS	1999/05/25	14: 48: 42	*USER	57344	*SYSTEM
5684	QEZRTBKS	QSYS	1999/05/25	14: 48: 17	*USER	40960	*SYSTEM
5685	QSYRAUTU	QSYS	1998/12/09	04: 29: 37	*USER	36864	*SYSTEM
5686	QSZCHKTG	QSYS	1998/12/08	13: 56: 28	*USER	32768	*I NHERI T
5687	QWDLSESE	QSYS	1999/01/13	11: 41: 38	*USER	73728	*SYSTEM
5688	QWTRTVPX	QSYS	1998/12/08	17: 39: 33	*USER	28672	*SYSTEM
5689	QWTSETPX	QSYS	1998/12/08	17: 40: 50	*USER	28672	*SYSTEM
5690	QGYRHRI	QSYS	1998/12/14	02: 38: 48	*USER	36864	*I NHERI T
5691	QGYRHRL	QSYS	1998/12/14	02: 38: 53	*USER	36864	*I NHERI T
5692	QWCRJBST	QSYS	1998/12/08	15: 17: 51	*OWNER	40960	*SYSTEM
5693	QSYCHGID	QSYS	1999/01/13	13: 42: 44	*USER	36864	*SYSTEM

Nbr	Name	Li b	Date	Time	Authori ty	Size	State
5694	QSYRTVUA	QSYS	1998/12/14	03: 21: 34	*USER	53248	*SYSTEM
5695	QSYCHGDS	QSYS	1998/12/08	16: 17: 54	*OWNER	45056	*SYSTEM
5696	QTADMPDV	QSYS	1998/12/09	04: 30: 13	*USER	225280	*SYSTEM
5697	QTARDI NF	QSYS	1998/12/09	04: 31: 46	*USER	45056	*SYSTEM
5698	QTQRCSC	QSYS	1998/12/08	14: 34: 03	*USER	20480	*I NHERI T
5699	QLGRLNGI	QSYS	1998/12/08	11: 38: 41	*USER	45056	*SYSTEM
5700	QTQSMXC2	QSYS	1998/12/08	14: 34: 09	*USER	20480	*I NHERI T
5701	QSYRTVSE	QSYS	1999/04/08	00: 52: 10	*USER	36864	*SYSTEM
5702	QWTCHGJB	QSYS	1999/05/14	14: 53: 29	*USER	270336	*SYSTEM
5703	QI MGCVTI	QSYS	1998/12/14	02: 56: 16	*USER	45056	*I NHERI T
5704	QSYADVLE	QSYS	1998/12/14	03: 01: 06	*USER	40960	*SYSTEM
5705	QSYCHVLE	QSYS	1998/12/14	03: 01: 11	*USER	40960	*SYSTEM
5706	QSYFDVLE	QSYS	1998/12/14	03: 01: 16	*USER	40960	*SYSTEM
5707	QSYRMVLE	QSYS	1998/12/14	03: 01: 21	*USER	36864	*SYSTEM
5708	QWCRI PLA	QSYS	1998/12/08	15: 15: 39	*USER	28672	*SYSTEM
5709	QWCRTVCA	QSYS	1998/12/14	03: 21: 45	*USER	98304	*SYSTEM
5710	QZLSADFS	QSYS	1999/04/13	12: 58: 12	*USER	94208	*SYSTEM
5711	QZLSADPS	QSYS	1999/04/13	12: 58: 16	*USER	86016	*SYSTEM
5712	QZLSCHFS	QSYS	1999/04/13	12: 58: 20	*USER	102400	*SYSTEM
5713	QZLSCHPS	QSYS	1999/04/13	12: 58: 24	*USER	69632	*SYSTEM
5714	QZLSCHSG	QSYS	1999/04/13	12: 58: 28	*USER	86016	*SYSTEM
5715	QZLSCHSI	QSYS	1999/04/13	12: 58: 31	*USER	86016	*SYSTEM
5716	QZLSCHSN	QSYS	1999/04/13	12: 58: 34	*USER	81920	*SYSTEM
5717	QZLESDS	QSYS	1999/04/13	12: 58: 38	*USER	53248	*SYSTEM
5718	QZLENSSS	QSYS	1999/04/13	12: 58: 41	*USER	57344	*SYSTEM
5719	QZLSLSTI	QSYS	1999/06/22	19: 54: 34	*USER	139264	*SYSTEM
5720	QZLSRMS	QSYS	1999/04/13	12: 58: 51	*USER	65536	*SYSTEM
5721	QZLSSTRS	QSYS	1999/04/13	12: 58: 54	*USER	57344	*SYSTEM
5722	QZLSOLST	QSYS	1999/04/13	12: 58: 48	*USER	69632	*SYSTEM
5723	QSYRUPWD	QSYS	1998/12/09	04: 29: 39	*USER	32768	*SYSTEM
5724	QSYSUPWD	QSYS	1999/08/25	10: 12: 15	*USER	40960	*SYSTEM
5725	QZNFRTVE	QSYS	1998/12/14	03: 24: 46	*USER	53248	*SYSTEM
5726	QSYADDUC	QSYS	1998/12/14	03: 20: 30	*USER	36864	*I NHERI T
5727	QSYADDVC	QSYS	1998/12/14	03: 20: 35	*USER	36864	*I NHERI T
5728	QSYCHKVC	QSYS	1998/12/14	03: 20: 45	*USER	36864	*I NHERI T
5729	QSYFNDUCU	QSYS	1998/12/14	03: 21: 00	*USER	36864	*I NHERI T
5730	QSYLSTUC	QSYS	1998/12/14	03: 21: 05	*USER	36864	*I NHERI T
5731	QSYLSTVC	QSYS	1998/12/14	03: 21: 10	*USER	36864	*I NHERI T
5732	QSYPARSC	QSYS	1998/12/14	03: 21: 15	*USER	36864	*I NHERI T
5733	QSYRMVUC	QSYS	1998/12/14	03: 21: 19	*USER	36864	*I NHERI T
5734	QSYRMVVC	QSYS	1998/12/14	03: 21: 24	*USER	36864	*I NHERI T
5735	QWTD TJBS	QSYS	1998/12/08	15: 31: 40	*USER	32768	*SYSTEM
5736	QLZAE LKE	QSYS	1998/12/08	00: 58: 00	*OWNER	40960	*SYSTEM
5737	QPOLFLOP	QSYS	1998/12/14	03: 16: 20	*OWNER	172032	*SYSTEM
5738	QQQCSDBM	QSYS	1999/04/06	12: 42: 08	*USER	40960	*SYSTEM
5739	QQQDSDBM	QSYS	1999/04/06	12: 42: 10	*USER	86016	*SYSTEM
5740	QQQESDBM	QSYS	1998/12/14	02: 42: 29	*USER	45056	*SYSTEM
5745	QQQTEMP1	QSYS	1999/05/27	13: 19: 43	*OWNER	86016	*SYSTEM
5746	QQQTEMP2	QSYS	1999/05/27	13: 19: 46	*OWNER	20480	*SYSTEM
5751	QTSPGETK	QSYS	1998/12/22	01: 11: 13	*USER	245760	*SYSTEM
5752	QTSPGETM	QSYS	1999/10/18	10: 46: 21	*USER	180224	*SYSTEM
5753	QTSPGETS	QSYS	1998/12/22	01: 12: 55	*USER	262144	*SYSTEM
5756	QTSPPUT	QSYS	1998/12/22	01: 13: 33	*USER	94208	*SYSTEM
5757	QTSPPUTM	QSYS	1998/12/22	01: 13: 57	*USER	90112	*SYSTEM
5758	QTSPUDR	QSYS	1998/12/22	01: 08: 40	*USER	139264	*SYSTEM
5805	QWC SVRD2	QSYS	1998/12/08	15: 17: 45	*USER	20480	*SYSTEM
5814	QFPAPRFJ	QSYS	1998/12/14	02: 35: 27	*OWNER	53248	*SYSTEM
5828	QPASVRP	QSYS	1998/12/08	12: 02: 39	*OWNER	49152	*SYSTEM
5829	QPASVRS	QSYS	1998/12/09	02: 44: 49	*OWNER	61440	*SYSTEM
5851	QSECSBD1	QSYS	1999/02/04	14: 17: 24	*USER	61440	*USER
5852	QSECSBD3	QSYS	1998/12/12	03: 48: 54	*USER	36864	*USER
5855	QSYPREAD	QSYS	1998/12/08	16: 21: 14	*USER	20480	*SYSTEM
5862	QTOCPPSM	QSYS	1999/10/05	14: 42: 56	*OWNER	65536	*SYSTEM
5865	QTOI SOPM	QSYS	1998/12/14	02: 32: 40	*USER	40960	*SYSTEM
5868	QXOTRCVP	QSYS	1998/12/08	17: 41: 01	*OWNER	16384	*SYSTEM
5874	QFPADAPP	QSYS	1998/12/08	08: 47: 25	*USER	24576	*SYSTEM
5898	QTOCRTTC	QSYS	1998/12/08	16: 45: 49	*USER	40960	*SYSTEM
5921	QBNVCSPE	QSYS	1998/12/08	06: 50: 24	*OWNER	20480	*SYSTEM
5925	QLGLI D	QSYS	1998/12/08	09: 50: 44	*USER	24576	*SYSTEM
5940	QJOADDRJ	QSYS	1998/12/08	11: 22: 47	*OWNER	61440	*SYSTEM
5942	QJOCHGST	QSYS	1998/12/08	09: 42: 00	*USER	98304	*SYSTEM
5949	QJORJRNI	QSYS	1998/12/08	11: 29: 09	*USER	32768	*SYSTEM

Nbr	Name	Li b	Date	Time	Authori ty	Size	State
5951	QJORMVRJ	QSYS	1998/12/08	09: 45: 57	*OWNER	45056	*SYSTEM
5952	QJORRCVI	QSYS	1998/12/08	11: 29: 33	*USER	28672	*SYSTEM
5968	QLSSPACE	QSYS	1998/11/17	20: 35: 30	*USER	16384	*I NHERI T
5980	QWCSCLXR	QSYS	1998/12/08	17: 22: 05	*OWNER	24576	*SYSTEM
5991	QJOPREAD	QSYS	1998/12/08	09: 43: 31	*USER	20480	*SYSTEM
6012	QTOCAUTO	QSYS	1999/05/14	13: 55: 00	*OWNER	32768	*SYSTEM
6015	QTOCI FCU	QSYS	1999/10/05	14: 41: 43	*OWNER	36864	*SYSTEM
6017	QTOCNETC	QSYS	1999/05/14	13: 54: 56	*OWNER	73728	*SYSTEM
6021	QTOCPPMU	QSYS	1999/10/05	14: 42: 35	*OWNER	114688	*SYSTEM
6022	QTOCPPPM	QSYS	1999/01/31	19: 55: 07	*OWNER	45056	*SYSTEM
6023	QTOCPPPU	QSYS	1999/07/21	06: 52: 16	*OWNER	69632	*SYSTEM
6024	QTOCRTEU	QSYS	1998/12/08	16: 44: 59	*OWNER	36864	*SYSTEM
6055	QTODDB2D	QSYS	1998/12/14	03: 11: 36	*OWNER	106496	*SYSTEM
6063	QXNAUPDF	QSYS	1999/09/01	10: 44: 19	*OWNER	77824	*SYSTEM
6064	QZSPI PI S	QSYS	1998/12/08	15: 55: 28	*OWNER	16384	*SYSTEM
6066	QTOCPORU	QSYS	1998/12/08	16: 44: 00	*OWNER	32768	*SYSTEM
6070	QYUNLANG	QSYS	1998/12/14	02: 49: 50	*USER	40960	*USER
6076	QFVCHVRM	QSYS	1998/12/08	07: 24: 39	*OWNER	45056	*SYSTEM
6081	QTEBRKWK	QSYS	1998/12/14	02: 54: 01	*USER	36864	*USER
6082	QTEDFI	QSYS	1998/12/14	02: 54: 03	*USER	45056	*I NHERI T
6087	QTEEVLWK	QSYS	1998/12/14	02: 54: 09	*USER	36864	*USER
6089	QTEMODN	QSYS	1998/12/14	02: 54: 13	*USER	61440	*SYSTEM
6090	QTEPGMWK	QSYS	1998/12/14	02: 54: 15	*USER	36864	*USER
6091	QTEPLSWK	QSYS	1998/12/14	02: 54: 16	*USER	36864	*USER
6092	QTESSH	QSYS	1999/02/19	19: 30: 28	*USER	53248	*USER
6093	QTESTOPH	QSYS	1999/02/19	19: 30: 30	*USER	45056	*USER
6099	QTEHDKW	QSYS	1998/12/14	02: 54: 33	*USER	36864	*USER
6100	QTEWCHWK	QSYS	1998/12/14	02: 54: 40	*USER	36864	*USER
6122	QWPLPTFL	QSYS	1999/09/01	13: 16: 04	*USER	94208	*SYSTEM
6125	QCPEXPRT	QSYS	1999/12/02	12: 58: 18	*USER	126976	*USER
6126	QCPI MPRT	QSYS	2000/01/26	13: 48: 22	*USER	172032	*USER
6131	QESI SRV	QSYS	1998/12/14	03: 02: 20	*OWNER	94208	*SYSTEM
6132	QESI STR	QSYS	1998/12/14	03: 02: 26	*OWNER	57344	*SYSTEM
6135	QESPHONE	QSYS	1998/12/12	04: 26: 20	*USER	40960	*USER
6138	QLZASS1X	QSYS	1998/11/17	18: 58: 52	*USER	20480	*USER
6140	QQQDBMWL	QSYS	1999/08/11	20: 41: 04	*USER	69632	*SYSTEM
6142	QSCBMKTE	QSYS	1998/12/14	22: 36: 52	*USER	45056	*SYSTEM
6143	QSCBPSP	QSYS	1998/12/14	22: 36: 53	*USER	36864	*SYSTEM
6144	QSCBSRVE	QSYS	1998/12/14	22: 36: 54	*USER	53248	*SYSTEM
6150	QSYRTCHI	QSYS	1998/12/08	13: 55: 20	*OWNER	32768	*SYSTEM
6160	QTVDMGR	QSYS	1998/12/08	14: 43: 06	*USER	20480	*SYSTEM
6161	QQQSSDBM	QSYS	1998/12/14	02: 43: 11	*USER	45056	*SYSTEM
6162	QSYCHFUI	QSYS	1998/12/14	03: 10: 52	*USER	40960	*I NHERI T
6163	QSYCKUFU	QSYS	1998/12/14	03: 10: 57	*USER	36864	*I NHERI T
6164	QSYDRGFN	QSYS	1998/12/14	03: 11: 07	*USER	36864	*I NHERI T
6165	QSYRGFN	QSYS	1998/12/14	03: 11: 17	*USER	36864	*I NHERI T
6166	QSYRTFUI	QSYS	1998/12/14	03: 11: 22	*USER	36864	*I NHERI T
6167	QSYRTUFI	QSYS	1998/12/14	03: 11: 26	*USER	36864	*I NHERI T
6168	QSYRTVFI	QSYS	1998/12/14	03: 11: 31	*USER	36864	*I NHERI T
6169	QTACJMA	QSYS	1999/04/24	10: 11: 52	*USER	53248	*SYSTEM
6170	QTARJMA	QSYS	1998/12/08	14: 06: 34	*USER	49152	*SYSTEM
6171	QWTSJUI D	QSYS	1998/12/14	03: 22: 10	*USER	40960	*SYSTEM
6172	QYPSENDS	QSYS	1999/01/26	15: 35: 23	*OWNER	49152	*SYSTEM
6173	QYPSSTRS	QSYS	1999/01/26	15: 35: 43	*OWNER	45056	*SYSTEM
6174	QHSMMOVF	QSYS	1998/12/08	10: 55: 09	*USER	28672	*SYSTEM
6175	QHSMMOVL	QSYS	1998/12/08	10: 53: 15	*USER	28672	*SYSTEM
6176	QQQQSDBM	QSYS	1998/12/14	02: 43: 04	*USER	40960	*SYSTEM
6177	QSRCRTMD	QSYS	1998/12/14	03: 04: 09	*USER	40960	*SYSTEM
6178	QSRDLTMD	QSYS	1998/12/14	03: 04: 15	*USER	40960	*SYSTEM
6179	QSRRTVMD	QSYS	1998/12/14	03: 04: 20	*USER	40960	*SYSTEM
6180	QWCRCLSI	QSYS	1998/12/08	17: 21: 34	*USER	28672	*SYSTEM
6181	QWDCSBSE	QSYS	1998/12/14	03: 22: 00	*USER	53248	*SYSTEM
6182	QSZSPTPR	QSYS	1999/01/29	23: 54: 57	*USER	77824	*SYSTEM
6183	QPZCPYSV	QSYS	1998/12/09	02: 45: 01	*OWNER	98304	*SYSTEM
6184	QSYDRGAP	QSYS	1998/12/14	03: 11: 02	*OWNER	36864	*I NHERI T
6185	QSYRGAP	QSYS	1998/12/14	03: 11: 12	*OWNER	36864	*I NHERI T
6186	QTACTLDV	QSYS	1998/12/08	14: 02: 46	*USER	57344	*SYSTEM
6187	QTOCLPPJ	QSYS	1998/12/14	03: 17: 37	*USER	139264	*SYSTEM
6189	QYPSCSA	QSYS	1999/01/26	15: 35: 08	*OWNER	36864	*I NHERI T
6190	QYPSCYCC	QSYS	1999/01/26	15: 35: 10	*OWNER	36864	*I NHERI T
6191	QYPSENDC	QSYS	1999/01/26	15: 35: 21	*OWNER	36864	*I NHERI T
6192	QYPSRSCA	QSYS	1999/01/26	15: 35: 30	*OWNER	36864	*I NHERI T

Nbr	Name	Li b	Date	Time	Authori ty	Size	State
6193	QYPSSTRC	QSYS	1999/01/26	15: 35: 40	*OWNER	36864	*INHERI T
6194	QYPSSTRT	QSYS	1999/01/26	15: 35: 45	*OWNER	36864	*INHERI T
6195	QSYRMVPT	QSYS	1998/12/14	03: 17: 26	*USER	40960	*SYSTEM
6231	QTVDMI EX	QSYS	1998/12/08	14: 43: 17	*USER	20480	*SYSTEM
6232	QTVENDTN	QSYS	1998/12/08	16: 57: 06	*USER	20480	*SYSTEM
6234	QTVSTRTN	QSYS	1998/12/08	16: 58: 26	*USER	24576	*SYSTEM
6235	QVTTELN	QSYS	1998/12/14	02: 48: 48	*USER	57344	*SYSTEM
6236	QUPMENU	QSYS	1998/11/18	03: 42: 37	*USER	8192	*USER
6237	QUPVLI C	QSYS	1998/11/18	03: 43: 05	*USER	16384	*USER
6254	QTOGI NTD	QSYS	1999/01/05	15: 40: 04	*USER	139264	*SYSTEM
6255	QTOGI NTE	QSYS	1998/12/14	02: 46: 46	*USER	49152	*USER
6263	QJODLTPH	QSYS	1998/12/08	07: 47: 09	*USER	28672	*SYSTEM
6264	QJORJRNE	QSYS	1998/12/08	11: 27: 55	*USER	45056	*SYSTEM
6289	QPOZPCP2	QSYS	1998/12/14	02: 55: 46	*USER	36864	*SYSTEM
6329	QZRUEDTF	QSYS	2000/01/18	15: 33: 54	*USER	299008	*USER
6338	QDCRNWI D	QSYS	1998/12/08	08: 46: 38	*USER	98304	*SYSTEM
6345	QQQUDFAC	QSYS	1999/08/24	19: 12: 30	*USER	36864	*USER
6349	QTOCL2TP	QSYS	1999/07/29	15: 49: 16	*USER	843776	*USER
6350	QTOCPPPCTL	QSYS	1999/06/24	21: 49: 29	*OWNER	196608	*SYSTEM
6354	QWTGETRQ	QSYS	1998/12/08	11: 41: 12	*USER	32768	*SYSTEM
6355	QWTPUTRQ	QSYS	1998/12/08	17: 39: 14	*USER	36864	*SYSTEM
6360	QYPSCDMC	QSYS	1999/01/26	15: 35: 01	*OWNER	176128	*SYSTEM
6365	QYSMCSVE	QSYS	1999/01/18	12: 31: 18	*OWNER	53248	*SYSTEM
6366	QYSMPUT	QSYS	1999/04/07	22: 09: 44	*USER	61440	*SYSTEM
6367	QYSMRSVE	QSYS	1998/12/14	03: 22: 50	*USER	45056	*SYSTEM

### Entries Sorted by Name

Nbr	Name	Lib	Date	Time	Authority	Size	State
3769	QAESTUB3	QSYS	1998/12/08	06: 20: 41	*USER	16384	*SYSTEM
5079	QALGENA	QSYS	1998/12/08	05: 42: 13	*USER	32768	*SYSTEM
5080	QALRTVA	QSYS	1998/12/08	06: 30: 19	*USER	36864	*SYSTEM
5081	QALSND	QSYS	1998/12/08	05: 43: 34	*USER	40960	*SYSTEM
5340	QBNLMODI	QSYS	1999/01/13	11: 35: 09	*USER	81920	*SYSTEM
5082	QBNLPGMI	QSYS	1999/01/13	11: 35: 30	*OWNER	122880	*SYSTEM
5083	QBNLSPGM	QSYS	1999/01/13	11: 35: 53	*OWNER	131072	*SYSTEM
5383	QBNRMODI	QSYS	1998/12/09	01: 47: 10	*USER	69632	*SYSTEM
5084	QBNRSPGM	QSYS	1998/12/08	05: 56: 18	*USER	49152	*SYSTEM
2494	QBNTGTRL	QSYS	1998/12/08	05: 57: 25	*USER	20480	*USER
5921	QBNVCSPE	QSYS	1998/12/08	06: 50: 24	*OWNER	20480	*SYSTEM
0030	QBSGET	QSYS	1998/12/08	06: 53: 12	*USER	69632	*SYSTEM
0039	QBSPUT	QSYS	1998/12/08	06: 00: 54	*USER	45056	*SYSTEM
2927	QACHECK	QSYS	1998/12/08	06: 00: 41	*USER	36864	*SYSTEM
0277	QCADRV	QSYS	1998/12/08	06: 01: 36	*USER	40960	*SYSTEM
0276	QCAEXEC	QSYS	1998/12/08	06: 54: 32	*USER	32768	*SYSTEM
5085	QCAPCMD	QSYS	1998/12/08	06: 02: 23	*USER	49152	*SYSTEM
3936	QCCWRKCC	QSYS	1998/12/08	06: 59: 25	*USER	57344	*SYSTEM
5086	QCDRCMDI	QSYS	1998/12/08	06: 56: 55	*USER	32768	*SYSTEM
1667	QCI ACCI N	QSYS	1998/12/08	06: 08: 14	*USER	57344	*SYSTEM
2127	QCI GETUL	QSYS	1998/12/08	06: 59: 52	*USER	20480	*SYSTEM
1666	QCI MRT	QSYS	1998/12/08	07: 06: 25	*USER	118784	*SYSTEM
1665	QCI OPEN	QSYS	1998/12/08	07: 03: 31	*USER	24576	*SYSTEM
3107	QCJOHATM	QSYS	1998/12/08	07: 09: 46	*USER	24576	*SYSTEM
2858	QCJONCI R	QSYS	1998/12/08	07: 08: 21	*USER	28672	*USER
2878	QCJQROUT	QSYS	1998/12/08	06: 17: 12	*USER	98304	*SYSTEM
0259	QCL	QSYS	1998/12/08	07: 11: 11	*USER	57344	*SYSTEM
0312	QCLCLCPR	QSYS	1998/12/08	07: 12: 11	*USER	61440	*SYSTEM
0314	QCLCLNUP	QSYS	1998/12/08	06: 17: 13	*USER	28672	*SYSTEM
0887	QCLCNVCN	QSYS	1998/12/08	07: 10: 57	*USER	24576	*I NHERI T
0888	QCLCNVNC	QSYS	1998/12/08	06: 17: 53	*USER	20480	*I NHERI T
0315	QCLDMI O	QSYS	1999/04/20	11: 51: 43	*USER	53248	*SYSTEM
2899	QCLRDTAQ	QSYS	1998/12/08	06: 21: 03	*USER	40960	*SYSTEM
5087	QCLRPGAS	QSYS	1998/12/08	06: 13: 21	*USER	36864	*SYSTEM
5088	QCLRPGMI	QSYS	1998/12/09	01: 52: 35	*USER	69632	*SYSTEM
0316	QCLRSLV	QSYS	1998/12/08	07: 18: 26	*USER	49152	*SYSTEM
0317	QCLRSVRE	QSYS	1998/12/08	06: 20: 38	*USER	24576	*SYSTEM
0313	QCLRTNE	QSYS	1998/12/08	06: 13: 13	*USER	20480	*SYSTEM
2926	QCLSCAN	QSYS	1998/12/08	06: 21: 21	*USER	24576	*I NHERI T
5089	QCLSPGAS	QSYS	1998/12/08	06: 14: 23	*USER	40960	*SYSTEM
0318	QCLSSVRE	QSYS	1998/12/08	07: 20: 18	*USER	28672	*SYSTEM
0889	QCLSWTCH	QSYS	1998/12/08	06: 14: 15	*USER	20480	*I NHERI T
0319	QCLXCXC	QSYS	1998/12/08	06: 22: 30	*USER	20480	*SYSTEM
0334	QCLXERR	QSYS	1998/12/08	06: 14: 24	*USER	24576	*SYSTEM
1962	QCMD	QSYS	1998/12/08	07: 25: 55	*USER	81920	*SYSTEM
1988	QCMDCHK	QSYS	1998/12/08	06: 22: 25	*USER	36864	*SYSTEM
1989	QCMDXC	QSYS	1998/12/08	06: 23: 20	*USER	32768	*SYSTEM
1418	QCOEXI T	QSYS	1998/12/12	04: 26: 18	*USER	28672	*USER
6125	QCPEXPRT	QSYS	1999/12/02	12: 58: 18	*USER	126976	*USER
6126	QCPI MPRT	QSYS	2000/01/26	13: 48: 22	*USER	172032	*USER
3069	QDBCHEOD	QSYS	1998/12/22	13: 39: 30	*USER	28672	*SYSTEM
0111	QDBFEOD	QSYS	1999/01/30	20: 15: 35	*USER	90112	*SYSTEM
0014	QDBGETDR	QSYS	1999/10/11	14: 39: 30	*USER	143360	*SYSTEM
0015	QDBGETKY	QSYS	1999/10/11	14: 36: 46	*USER	155648	*SYSTEM
0770	QDBGETM	QSYS	1999/10/18	10: 57: 04	*USER	196608	*SYSTEM
0016	QDBGETSQ	QSYS	1999/10/11	14: 35: 49	*USER	151552	*SYSTEM
5403	QDBLDBR	QSYS	1999/01/13	11: 36: 12	*OWNER	73728	*SYSTEM
4301	QDBLNGMV	QSYS	1998/12/08	06: 23: 25	*USER	28672	*I NHERI T
0018	QDBPUT	QSYS	1999/08/25	13: 26: 17	*USER	139264	*SYSTEM
1678	QDBPUTD	QSYS	1998/12/22	13: 38: 29	*USER	32768	*SYSTEM
1712	QDBPUTDR	QSYS	1999/08/25	13: 25: 37	*USER	139264	*SYSTEM
0768	QDBPUTM	QSYS	1999/01/30	20: 23: 19	*USER	90112	*SYSTEM
5092	QDBRTVFD	QSYS	1999/05/21	10: 46: 22	*USER	81920	*SYSTEM
5678	QDBRTVSN	QSYS	1999/05/27	18: 42: 04	*OWNER	57344	*SYSTEM
0019	QDBUDR	QSYS	1999/08/25	13: 24: 58	*USER	167936	*SYSTEM
5436	QDCCCFGD	QSYS	1998/12/08	06: 56: 35	*USER	45056	*SYSTEM
5095	QDCLCFGD	QSYS	1999/01/13	11: 36: 21	*USER	61440	*SYSTEM
4958	QDCPOLP	QSYS	1998/12/08	06: 41: 22	*USER	20480	*USER
5096	QDCRCFGS	QSYS	1998/12/08	07: 30: 13	*USER	49152	*SYSTEM

Nbr	Name	Li b	Date	Time	Authori ty	Si ze	State
5097	QDCRCTL D	QSYS	1998/12/08	07: 32: 37	*USER	192512	*SYSTEM
5098	QDCRDEVD	QSYS	1999/03/25	18: 04: 29	*USER	221184	*SYSTEM
5099	QDCRLI ND	QSYS	1999/03/30	18: 07: 53	*USER	438272	*SYSTEM
6338	QDCRNWI D	QSYS	1998/12/08	08: 46: 38	*USER	98304	*SYSTEM
4926	QDCRNWSD	QSYS	1998/12/08	08: 46: 46	*USER	69632	*SYSTEM
0410	QDCXLATE	QSYS	1998/12/08	08: 57: 08	*USER	61440	*SYSTEM
5100	QDFRTVFD	QSYS	1998/12/08	07: 49: 58	*USER	40960	*SYSTEM
0097	QDKFEOD	QSYS	1998/12/08	09: 10: 31	*USER	24576	*SYSTEM
0060	QDKGET	QSYS	1998/12/08	07: 50: 25	*USER	28672	*SYSTEM
0947	QDKGETI	QSYS	1998/12/08	07: 50: 02	*USER	28672	*SYSTEM
0061	QDKPUT	QSYS	1998/12/08	09: 10: 40	*USER	28672	*SYSTEM
0946	QDKPUTI	QSYS	1998/12/08	07: 50: 22	*USER	28672	*SYSTEM
0121	QDMACCI N	QSYS	1998/12/08	06: 54: 44	*USER	32768	*SYSTEM
0066	QDMACQDV	QSYS	1998/12/09	02: 23: 02	*USER	73728	*SYSTEM
0011	QDMCLOSE	QSYS	1998/12/08	09: 13: 27	*USER	49152	*SYSTEM
0012	QDMCOPEN	QSYS	1999/04/20	11: 53: 39	*USER	180224	*SYSTEM
1951	QDMGETST	QSYS	1998/12/08	09: 17: 17	*USER	57344	*SYSTEM
0069	QDMI FERR	QSYS	1998/12/08	07: 53: 07	*USER	32768	*SYSTEM
0063	QDMNODEV	QSYS	1998/12/08	09: 15: 23	*USER	28672	*SYSTEM
0071	QDMRLSDV	QSYS	1998/12/08	09: 17: 53	*USER	32768	*SYSTEM
4883	QDMRTVFO	QSYS	1999/04/02	12: 36: 28	*USER	32768	*SYSTEM
2767	QDXHRTR	QSYS	1998/12/08	09: 24: 27	*OWNER	102400	*SYSTEM
4097	QDOENDTS	QSYS	1998/12/08	08: 02: 43	*USER	32768	*SYSTEM
4096	QDOSTRTS	QSYS	1999/05/25	14: 52: 07	*USER	53248	*SYSTEM
4099	QDOTRND S	QSYS	1998/12/08	09: 24: 31	*USER	45056	*SYSTEM
0757	QEARBKM	QSYS	1998/12/08	06: 59: 05	*USER	24576	*SYSTEM
5101	QEARMVBM	QSYS	1998/12/08	09: 26: 50	*USER	32768	*SYSTEM
3896	QECCVTEC	QSYS	1998/12/08	09: 28: 20	*OWNER	32768	*SYSTEM
3897	QECCVTEW	QSYS	1998/12/08	08: 04: 09	*USER	28672	*I NHERI T
0411	QECEDI TW	QSYS	1998/12/08	09: 28: 16	*USER	28672	*I NHERI T
3898	QECEDT	QSYS	1998/12/08	08: 05: 08	*USER	28672	*I NHERI T
4556	QEMCFGI N	QSYS	1998/12/08	09: 34: 33	*USER	24576	*SYSTEM
1196	QEMPEBSC	QSYS	1999/05/25	14: 51: 11	*USER	69632	*SYSTEM
1197	QEMPESNA	QSYS	1999/05/25	14: 51: 50	*USER	102400	*SYSTEM
5276	QESANOTE	QSYS	1998/12/08	08: 12: 17	*OWNER	32768	*SYSTEM
4833	QESCSRVA	QSYS	1998/12/08	09: 41: 28	*USER	36864	*SYSTEM
4829	QESDSPSA	QSYS	1998/12/08	08: 14: 48	*OWNER	24576	*SYSTEM
6131	QESI SRV	QSYS	1998/12/14	03: 02: 20	*OWNER	94208	*SYSTEM
6132	QESI STR	QSYS	1998/12/14	03: 02: 26	*OWNER	57344	*SYSTEM
6135	QESPHONE	QSYS	1998/12/12	04: 26: 20	*USER	40960	*USER
4832	QESRSRVA	QSYS	1998/12/08	09: 43: 50	*OWNER	36864	*SYSTEM
4834	QESSAVMA	QSYS	1998/12/08	09: 42: 45	*USER	20480	*SYSTEM
4345	QEXCHJCB	QSYS	1998/12/08	08: 18: 42	*USER	20480	*SYSTEM
3748	QEXCLRCI	QSYS	1998/12/08	09: 50: 10	*USER	16384	*SYSTEM
2796	QEXFPGM	QSYS	1998/12/08	08: 26: 53	*USER	24576	*SYSTEM
5412	QEZAST	QSYS	1998/12/12	04: 26: 22	*USER	28672	*USER
2894	QEZATNWD	QSYS	1998/12/08	08: 31: 36	*USER	24576	*SYSTEM
5413	QEZBCHJB	QSYS	1998/12/12	04: 26: 23	*USER	28672	*USER
3241	QEZBCKUP	QSYS	1998/12/08	09: 58: 46	*USER	73728	*SYSTEM
3916	QEZBKMSG	QSYS	1998/12/08	08: 31: 17	*USER	24576	*SYSTEM
3915	QEZBKSCD	QSYS	1998/12/08	10: 00: 45	*USER	61440	*SYSTEM
3917	QEZBKWM	QSYS	1998/12/08	08: 31: 23	*USER	20480	*SYSTEM
5679	QEZCHBKL	QSYS	1998/12/08	08: 32: 33	*USER	40960	*SYSTEM
5680	QEZCHBKS	QSYS	1999/05/25	14: 48: 59	*USER	73728	*SYSTEM
3700	QEZCHGOP	QSYS	1998/12/08	07: 12: 51	*USER	40960	*SYSTEM
3701	QEZCHGPW	QSYS	1998/12/08	10: 00: 33	*USER	28672	*SYSTEM
3633	QEZCLINT	QSYS	1998/12/08	10: 03: 38	*OWNER	36864	*SYSTEM
2893	QEZCNPWR	QSYS	1998/12/08	08: 35: 13	*USER	28672	*SYSTEM
3247	QEZINIT	QSYS	1998/12/08	10: 05: 12	*USER	40960	*SYSTEM
5102	QEZLSGNU	QSYS	1999/01/13	11: 36: 34	*USER	77824	*SYSTEM
2896	QEZMAI N	QSYS	1998/12/08	10: 06: 52	*USER	20480	*SYSTEM
5414	QEZMSG	QSYS	1998/12/12	04: 26: 34	*USER	28672	*USER
5415	QEZOUTPT	QSYS	1998/12/12	04: 26: 34	*USER	28672	*USER
3206	QEZPRDEV	QSYS	1998/12/08	10: 07: 17	*USER	20480	*SYSTEM
5681	QEZRTBKD	QSYS	1999/05/25	14: 48: 24	*USER	45056	*SYSTEM
5682	QEZRTBKH	QSYS	1999/05/25	14: 48: 29	*USER	36864	*SYSTEM
5683	QEZRTBKO	QSYS	1999/05/25	14: 48: 42	*USER	57344	*SYSTEM
5684	QEZRTBKS	QSYS	1999/05/25	14: 48: 17	*USER	40960	*SYSTEM
3209	QEZSAVIN	QSYS	1998/12/08	10: 09: 26	*USER	32768	*SYSTEM
4252	QEZSNDMG	QSYS	1998/12/08	08: 40: 10	*USER	40960	*SYSTEM
4169	QEZSVI BM	QSYS	1998/12/08	08: 41: 14	*USER	28672	*SYSTEM
1900	QEZWRCLU	QSYS	1998/12/08	08: 41: 31	*OWNER	32768	*SYSTEM

Nbr	Name	Li b	Date	Time	Authori ty	Si ze	State
4207	QEZWRDSK	QSYS	1998/12/08	07: 18: 10	*USER	57344	*SYSTEM
1257	QFNREAD	QSYS	1998/12/08	07: 18: 40	*USER	20480	*SYSTEM
1550	QFNREADI	QSYS	1998/12/08	10: 15: 48	*USER	20480	*SYSTEM
1255	QFNROUTE	QSYS	1998/12/08	10: 18: 03	*USER	65536	*SYSTEM
1258	QFNWRT	QSYS	1998/12/08	08: 45: 29	*USER	20480	*SYSTEM
1259	QFNWRTI	QSYS	1998/12/08	07: 19: 10	*USER	20480	*SYSTEM
4791	QFPAACTV	QSYS	1998/12/14	02: 34: 49	*OWNER	40960	*SYSTEM
5874	QFPADAPP	QSYS	1998/12/08	08: 47: 25	*USER	24576	*SYSTEM
4792	QFPAMON	QSYS	2000/01/17	08: 22: 33	*OWNER	98304	*SYSTEM
5661	QFPAMONB	QSYS	1998/12/14	02: 35: 11	*OWNER	102400	*SYSTEM
5662	QFPAMONN	QSYS	1998/12/14	02: 35: 16	*OWNER	114688	*SYSTEM
5814	QFPAPRFJ	QSYS	1998/12/14	02: 35: 27	*OWNER	53248	*SYSTEM
4888	QFPHDLSS	QSYS	1998/12/08	07: 23: 52	*OWNER	24576	*SYSTEM
2848	QFSMAI N	QSYS	1998/12/08	10: 31: 05	*USER	24576	*SYSTEM
4540	QFVADDA	QSYS	1998/12/08	08: 56: 51	*USER	102400	*SYSTEM
6076	QFVCHVRM	QSYS	1998/12/08	07: 24: 39	*OWNER	45056	*SYSTEM
4545	QFVLSTA	QSYS	1999/01/13	11: 36: 55	*USER	102400	*SYSTEM
4332	QFVLSTNL	QSYS	1999/01/13	11: 37: 09	*USER	77824	*SYSTEM
4546	QFVRMVA	QSYS	1998/12/08	08: 57: 54	*USER	57344	*SYSTEM
4547	QFVRTVCD	QSYS	1998/12/08	07: 26: 15	*USER	69632	*SYSTEM
1265	QGDACO	QSYS	1998/12/08	10: 31: 59	*USER	57344	*SYSTEM
1267	QGDAL NVP	QSYS	1998/12/08	08: 58: 36	*USER	20480	*SYSTEM
5438	QGSCPYRS	QSYS	1999/01/13	11: 37: 18	*USER	65536	*SYSTEM
5437	QGSLRSC	QSYS	1999/01/13	11: 38: 31	*OWNER	200704	*SYSTEM
5690	QGYRHRI	QSYS	1998/12/14	02: 38: 48	*USER	36864	*I NHERI T
5691	QGYRHRL	QSYS	1998/12/14	02: 38: 53	*USER	36864	*I NHERI T
3671	QHCCCFG	QSYS	1998/12/08	10: 50: 51	*USER	114688	*SYSTEM
3298	QHCORYLN	QSYS	1998/12/08	09: 10: 16	*USER	24576	*SYSTEM
3983	QHFCGAT	QSYS	1998/12/08	09: 11: 56	*USER	53248	*SYSTEM
3980	QHFCGFP	QSYS	1998/12/08	07: 32: 34	*USER	36864	*SYSTEM
5103	QHFCLODR	QSYS	1998/12/08	10: 51: 26	*USER	36864	*SYSTEM
3981	QHFCLOSF	QSYS	1998/12/08	10: 51: 27	*USER	36864	*SYSTEM
5104	QHFCPYSF	QSYS	1998/12/08	09: 11: 11	*USER	61440	*SYSTEM
5105	QHFCRTDR	QSYS	1998/12/08	09: 11: 40	*USER	53248	*SYSTEM
5106	QHFCRLF	QSYS	1998/12/08	07: 34: 04	*USER	53248	*SYSTEM
5431	QHFDLTR	QSYS	1998/12/08	10: 51: 50	*USER	45056	*SYSTEM
3984	QHFDLTFS	QSYS	1998/12/08	10: 51: 17	*USER	45056	*SYSTEM
5422	QHFDRGFS	QSYS	1998/12/08	09: 11: 58	*USER	40960	*SYSTEM
5107	QHFFRCFS	QSYS	1998/12/08	10: 52: 05	*USER	36864	*SYSTEM
5108	QHFGTSZ	QSYS	1998/12/08	09: 13: 40	*USER	36864	*SYSTEM
5109	QHFLSTFS	QSYS	1999/01/13	11: 38: 41	*USER	53248	*SYSTEM
5110	QHFLULSF	QSYS	1998/12/08	10: 51: 53	*USER	36864	*SYSTEM
5321	QHFMOVSF	QSYS	1998/12/08	09: 12: 58	*USER	61440	*SYSTEM
5322	QHFOPNDR	QSYS	1998/12/08	09: 12: 59	*USER	53248	*SYSTEM
3977	QHFOPNFS	QSYS	1998/12/08	07: 34: 43	*USER	53248	*SYSTEM
5323	QHFRDDR	QSYS	1998/12/08	10: 53: 26	*USER	40960	*SYSTEM
3978	QHFRDSF	QSYS	1998/12/08	10: 54: 19	*USER	40960	*SYSTEM
5324	QHFRGFS	QSYS	1998/12/08	09: 13: 05	*USER	49152	*SYSTEM
5325	QHFRNMDR	QSYS	1998/12/08	09: 12: 42	*USER	49152	*SYSTEM
5326	QHFRNMSF	QSYS	1998/12/08	07: 34: 33	*USER	49152	*SYSTEM
3982	QHFRTVAT	QSYS	1998/12/08	10: 54: 35	*USER	57344	*SYSTEM
5327	QHFSZTSZ	QSYS	1998/12/08	09: 15: 05	*USER	36864	*SYSTEM
3979	QHFWRTSF	QSYS	1998/12/08	09: 13: 18	*USER	40960	*SYSTEM
6174	QHSMMOV	QSYS	1998/12/08	10: 55: 09	*USER	28672	*SYSTEM
6175	QHSMMOVL	QSYS	1998/12/08	10: 53: 15	*USER	28672	*SYSTEM
1698	QI CGET	QSYS	1999/01/07	14: 33: 16	*USER	73728	*SYSTEM
1699	QI CPUT	QSYS	1998/12/08	09: 17: 50	*USER	151552	*SYSTEM
5703	QI MGCVTI	QSYS	1998/12/14	02: 56: 16	*USER	45056	*I NHERI T
5940	QJOADDRJ	QSYS	1998/12/08	11: 22: 47	*OWNER	61440	*SYSTEM
5942	QJOCHGST	QSYS	1998/12/08	09: 42: 00	*USER	98304	*SYSTEM
6263	QJODLTPH	QSYS	1998/12/08	07: 47: 09	*USER	28672	*SYSTEM
5991	QJOPREAD	QSYS	1998/12/08	09: 43: 31	*USER	20480	*SYSTEM
5423	QJORJIDI	QSYS	1998/12/08	09: 45: 25	*USER	49152	*SYSTEM
6264	QJORJRNE	QSYS	1998/12/08	11: 27: 55	*USER	45056	*SYSTEM
5949	QJORJRNI	QSYS	1998/12/08	11: 29: 09	*USER	32768	*SYSTEM
5951	QJORMVRJ	QSYS	1998/12/08	09: 45: 57	*OWNER	45056	*SYSTEM
5952	QJORRCVI	QSYS	1998/12/08	11: 29: 33	*USER	28672	*SYSTEM
4879	QJOSJRNE	QSYS	1998/12/08	07: 50: 41	*USER	36864	*SYSTEM
4463	QLEDAGE	QSYS	1998/12/14	03: 25: 08	*USER	118784	*SYSTEM
5063	QLGCNVCS	QSYS	1998/12/14	02: 33: 00	*USER	36864	*I NHERI T
4175	QLGCNVSS	QSYS	1998/12/08	07: 51: 49	*USER	32768	*SYSTEM
5925	QLGGLI D	QSYS	1998/12/08	09: 50: 44	*USER	24576	*SYSTEM



Nbr	Name	Li b	Date	Time	Authori ty	Si ze	State
5699	QLGRLNGI	QSYS	1998/12/08	11: 38: 41	*USER	45056	*SYSTEM
5441	QLGRTVLC	QSYS	1998/12/14	03: 02: 41	*USER	40960	*INHERI T
5328	QLGRTVLI	QSYS	1998/12/08	09: 51: 55	*USER	32768	*SYSTEM
4171	QLGRTVSS	QSYS	1999/03/03	18: 38: 47	*USER	53248	*SYSTEM
5329	QLGSCNMX	QSYS	1998/12/08	07: 52: 34	*USER	28672	*SYSTEM
5330	QLGSORT	QSYS	1998/12/08	11: 37: 27	*USER	57344	*SYSTEM
5331	QLGSRTI O	QSYS	1998/12/09	02: 27: 37	*USER	143360	*SYSTEM
4981	QLGTRDTA	QSYS	1998/12/08	09: 52: 13	*USER	36864	*SYSTEM
4173	QLGVLI D	QSYS	1998/12/08	07: 53: 03	*USER	28672	*SYSTEM
4946	QLI CHGLL	QSYS	1998/12/08	09: 53: 26	*USER	32768	*SYSTEM
0074	QLI CNV	QSYS	1998/12/08	09: 53: 48	*USER	28672	*SYSTEM
4553	QLI COBJD	QSYS	1998/12/08	07: 56: 16	*USER	57344	*SYSTEM
4552	QLI CVTTP	QSYS	1998/12/08	07: 53: 33	*USER	32768	*SYSTEM
4947	QLI RLI BD	QSYS	1998/12/08	07: 55: 42	*USER	40960	*SYSTEM
4948	QLI RNMO	QSYS	1998/12/09	02: 44: 51	*USER	65536	*SYSTEM
0095	QLPBATI N	QSYS	1999/02/18	22: 09: 13	*OWNER	77824	*SYSTEM
5439	QLPCDI NF	QSYS	1998/12/14	03: 02: 52	*USER	36864	*INHERI T
5440	QLPCDRST	QSYS	1998/12/14	03: 02: 57	*USER	36864	*INHERI T
3474	QLPCRTDT	QSYS	1998/12/08	10: 01: 25	*USER	45056	*SYSTEM
3469	QLPINATO	QSYS	1999/05/18	19: 37: 34	*OWNER	262144	*SYSTEM
3471	QLPINLPP	QSYS	1999/05/18	19: 41: 56	*OWNER	393216	*SYSTEM
5456	QLPI SLNG	QSYS	1998/12/08	08: 01: 02	*USER	40960	*SYSTEM
3197	QLPI VPD	QSYS	1998/12/08	11: 53: 04	*USER	28672	*SYSTEM
5332	QLPLPRDS	QSYS	1999/01/13	11: 34: 56	*USER	69632	*SYSTEM
3879	QLPRMPGM	QSYS	1999/01/29	23: 23: 23	*USER	49152	*USER
3470	QLPWRKI P	QSYS	1998/12/08	08: 02: 44	*USER	40960	*SYSTEM
5416	QLRCHGCM	QSYS	1998/12/09	20: 16: 36	*USER	28672	*INHERI T
5417	QLRRTVCE	QSYS	1998/12/09	20: 17: 00	*USER	28672	*INHERI T
5418	QLRSETCE	QSYS	1998/12/09	20: 13: 52	*USER	36864	*INHERI T
5968	QLSSPACE	QSYS	1998/11/17	20: 35: 30	*USER	16384	*INHERI T
4043	QLYGETS	QSYS	1998/12/08	08: 02: 18	*USER	20480	*SYSTEM
4045	QLYRDBI	QSYS	1998/12/08	11: 58: 48	*USER	24576	*SYSTEM
4042	QLYSETS	QSYS	1998/12/08	11: 58: 59	*USER	24576	*SYSTEM
4044	QLYWRTBI	QSYS	1998/12/08	10: 07: 05	*USER	24576	*SYSTEM
5333	QLZAADDK	QSYS	1998/12/08	08: 03: 18	*USER	32768	*SYSTEM
5334	QLZADDLI	QSYS	1998/12/08	11: 59: 48	*OWNER	53248	*SYSTEM
5736	QLZAEKE	QSYS	1998/12/08	00: 58: 00	*OWNER	40960	*SYSTEM
5419	QLZAGENK	QSYS	1998/12/08	10: 08: 04	*OWNER	49152	*SYSTEM
4459	QLZAREQ	QSYS	1999/01/27	17: 57: 20	*OWNER	90112	*SYSTEM
4460	QLZARLS	QSYS	1998/12/08	10: 09: 21	*OWNER	53248	*SYSTEM
4461	QLZARTV	QSYS	1998/12/08	12: 03: 28	*USER	36864	*SYSTEM
5335	QLZARTVK	QSYS	1998/12/08	10: 09: 36	*USER	32768	*SYSTEM
6138	QLZASS1X	QSYS	1998/11/17	18: 58: 52	*USER	20480	*USER
5421	QMARQSOA	QSYS	1998/12/08	10: 14: 20	*OWNER	36864	*SYSTEM
4835	QMHCHGEM	QSYS	1999/08/16	14: 54: 52	*USER	36864	*SYSTEM
5420	QMHCTLJL	QSYS	1998/12/08	10: 17: 01	*USER	49152	*SYSTEM
4836	QMHJJOBL	QSYS	1999/01/13	11: 33: 38	*OWNER	155648	*SYSTEM
4837	QMHLMSTM	QSYS	1999/01/13	11: 34: 11	*USER	159744	*SYSTEM
4265	QMHMOVPM	QSYS	1999/08/16	14: 54: 21	*USER	81920	*SYSTEM
4838	QMHPRMM	QSYS	1998/12/08	10: 20: 58	*USER	45056	*SYSTEM
3210	QMHORDQD	QSYS	1999/04/28	12: 09: 11	*USER	32768	*SYSTEM
4263	QMHRCVM	QSYS	1999/06/14	16: 59: 14	*USER	143360	*SYSTEM
4264	QMHRCVPM	QSYS	1999/08/16	15: 08: 21	*USER	184320	*SYSTEM
4878	QMHREQM	QSYS	1999/04/28	12: 09: 20	*USER	45056	*SYSTEM
4842	QMHMFAT	QSYS	1998/12/08	10: 22: 39	*USER	32768	*SYSTEM
4840	QMHMQAT	QSYS	1998/12/08	08: 15: 13	*USER	40960	*SYSTEM
4261	QMHMMVM	QSYS	1998/12/08	12: 27: 54	*USER	65536	*SYSTEM
4262	QMHMVPM	QSYS	1999/08/16	15: 08: 40	*USER	77824	*SYSTEM
4266	QMHRSNEM	QSYS	1999/08/16	14: 54: 47	*USER	81920	*SYSTEM
4839	QMHRTVM	QSYS	1998/12/08	08: 14: 16	*USER	40960	*SYSTEM
2445	QMHRTVRQ	QSYS	1998/12/08	12: 30: 32	*USER	36864	*SYSTEM
4841	QMHSNDBM	QSYS	1998/12/08	12: 27: 49	*USER	32768	*SYSTEM
4268	QMHSNDM	QSYS	1998/12/08	10: 23: 28	*USER	36864	*SYSTEM
4269	QMHNDPM	QSYS	1999/08/16	15: 09: 17	*USER	131072	*SYSTEM
4270	QMHNSDRM	QSYS	1998/12/08	08: 16: 30	*USER	151552	*SYSTEM
5336	QMHNSDSM	QSYS	1999/08/16	14: 53: 59	*USER	36864	*SYSTEM
2856	QMNHCMD	QSYS	1998/12/08	10: 31: 50	*USER	45056	*SYSTEM
2357	QMNBSBS	QSYS	1998/12/08	08: 15: 59	*USER	20480	*SYSTEM
2358	QMNUI M	QSYS	1999/08/01	17: 22: 26	*USER	36864	*SYSTEM
5337	QNMCHGMN	QSYS	1998/12/09	02: 27: 24	*USER	32768	*SYSTEM
5338	QNMDRGAP	QSYS	1998/12/09	02: 34: 09	*USER	32768	*SYSTEM
5339	QNMDRGFN	QSYS	1998/12/08	12: 46: 25	*OWNER	40960	*SYSTEM

Nbr	Name	Li b	Date	Time	Authori ty	Si ze	State
5341	QNMDRGTI	QSYS	1998/12/09	02: 34: 11	*USER	32768	*SYSTEM
5342	QNMENDAP	QSYS	1998/12/09	02: 55: 12	*USER	32768	*SYSTEM
5343	QNMRCVDT	QSYS	1998/12/09	02: 35: 07	*USER	45056	*SYSTEM
5344	QNMRCVOC	QSYS	1998/12/09	02: 35: 06	*USER	32768	*SYSTEM
5345	QNMREGAP	QSYS	1998/12/09	02: 55: 29	*USER	32768	*SYSTEM
5346	QNMRGFN	QSYS	1998/12/08	12: 48: 43	*OWNER	45056	*SYSTEM
5347	QNMRTI	QSYS	1999/01/13	11: 34: 24	*USER	73728	*SYSTEM
5348	QNMRRGF	QSYS	1998/12/08	10: 40: 28	*OWNER	40960	*SYSTEM
5349	QNMRTVMN	QSYS	1998/12/09	02: 28: 22	*USER	28672	*SYSTEM
5350	QNMSENDER	QSYS	1998/12/09	02: 35: 06	*USER	32768	*SYSTEM
5351	QNMSENDERP	QSYS	1998/12/09	02: 36: 07	*USER	40960	*SYSTEM
5352	QNMSENDERQ	QSYS	1998/12/09	02: 56: 35	*USER	40960	*SYSTEM
5353	QNMSTRAP	QSYS	1998/12/09	02: 56: 58	*USER	36864	*SYSTEM
4029	QOCCTLOF	QSYS	1998/12/08	10: 46: 41	*USER	20480	*SYSTEM
3193	QOEB13	QSYS	1998/11/17	21: 34: 16	*USER	8192	*SYSTEM
3194	QOEB14	QSYS	1998/11/17	19: 24: 34	*USER	8192	*SYSTEM
3195	QOEB15	QSYS	1998/11/17	22: 57: 49	*USER	8192	*SYSTEM
3196	QOEB16	QSYS	1998/11/17	22: 58: 13	*USER	8192	*SYSTEM
2421	QOECMPRT	QSYS	1999/06/04	13: 33: 28	*USER	36864	*SYSTEM
5354	QOGCHGOE	QSYS	1998/12/08	13: 25: 02	*OWNER	49152	*SYSTEM
5355	QOGRVVOE	QSYS	1998/12/08	11: 08: 29	*OWNER	36864	*SYSTEM
2180	QOHFI XI X	QSYS	1998/12/08	13: 23: 55	*OWNER	20480	*SYSTEM
3745	QOHFI XQ	QSYS	1998/12/08	13: 24: 33	*OWNER	24576	*SYSTEM
5671	QOKADDDP	QSYS	1998/12/08	11: 12: 55	*OWNER	28672	*SYSTEM
4014	QOKCCTL	QSYS	1998/12/08	13: 32: 24	*OWNER	61440	*SYSTEM
4020	QOKCCTOR	QSYS	1998/12/08	11: 15: 41	*OWNER	192512	*SYSTEM
5672	QOKCHGDP	QSYS	1998/12/08	13: 29: 35	*OWNER	28672	*SYSTEM
4023	QOKDSPDP	QSYS	1998/12/08	08: 53: 15	*OWNER	28672	*SYSTEM
5356	QOKDSPX4	QSYS	1998/12/08	13: 31: 22	*OWNER	32768	*SYSTEM
5673	QOKRMVDP	QSYS	1998/12/08	13: 35: 21	*OWNER	28672	*SYSTEM
4885	QOKSCHD	QSYS	1998/12/08	08: 57: 45	*OWNER	229376	*SYSTEM
2182	QOLDLI NK	QSYS	1998/12/08	13: 37: 39	*USER	45056	*SYSTEM
2181	QOLELI NK	QSYS	1998/12/08	11: 25: 57	*OWNER	73728	*SYSTEM
2189	QOLOLI ND	QSYS	1998/12/08	13: 39: 23	*USER	40960	*SYSTEM
2187	QOLRECV	QSYS	1998/12/08	11: 23: 46	*OWNER	40960	*SYSTEM
2186	QOLSEND	QSYS	1998/12/08	11: 23: 58	*OWNER	45056	*SYSTEM
2188	QOLSETF	QSYS	1998/12/08	13: 37: 57	*USER	32768	*SYSTEM
2211	QOLTI MER	QSYS	1998/12/08	13: 38: 22	*USER	28672	*SYSTEM
1605	QOSPRI NT	QSYS	1998/12/08	11: 48: 27	*OWNER	24576	*SYSTEM
5828	QPASVRP	QSYS	1998/12/08	12: 02: 39	*OWNER	49152	*SYSTEM
5829	QPASVRS	QSYS	1998/12/09	02: 44: 49	*OWNER	61440	*SYSTEM
5040	QPDOUTP2	QSYS	1998/12/08	14: 16: 28	*OWNER	20480	*SYSTEM
5044	QPDGENSS	QSYS	1998/12/14	02: 30: 40	*USER	73728	*SYSTEM
4149	QPDLOGER	QSYS	1998/12/08	14: 16: 28	*USER	40960	*SYSTEM
4259	QPDWRKPB	QSYS	1998/12/08	14: 19: 07	*USER	24576	*SYSTEM
0309	QPGMMENU	QSYS	1998/12/08	12: 08: 18	*USER	143360	*SYSTEM
4398	QPMACLCT	QSYS	1998/12/09	02: 45: 01	*OWNER	147456	*SYSTEM
4397	QPMASERV	QSYS	1998/12/08	14: 19: 42	*OWNER	86016	*SYSTEM
4323	QPMBPCS	QSYS	1998/11/18	02: 21: 05	*OWNER	8192	*SYSTEM
5357	QPMPLFRD	QSYS	1998/12/08	14: 22: 18	*USER	36864	*SYSTEM
5358	QPMWKCCL	QSYS	1999/08/17	16: 55: 34	*OWNER	53248	*SYSTEM
3008	QPOPDMGR	QSYS	1998/12/09	03: 14: 09	*OWNER	356352	*SYSTEM
5359	QPRCRTPG	QSYS	1998/12/08	12: 19: 18	*USER	49152	*SYSTEM
6183	QPZCPYSV	QSYS	1998/12/09	02: 45: 01	*OWNER	98304	*SYSTEM
5360	QPZCRTFX	QSYS	1998/12/08	14: 36: 47	*USER	73728	*SYSTEM
3622	QPZDLOBJ	QSYS	1998/12/08	09: 34: 22	*OWNER	61440	*SYSTEM
5361	QPZGENNM	QSYS	1998/12/08	14: 37: 06	*USER	36864	*SYSTEM
5362	QPZLOGFX	QSYS	1998/12/08	12: 24: 23	*USER	36864	*SYSTEM
4746	QPZPTFI N	QSYS	1998/12/09	03: 12: 31	*OWNER	77824	*SYSTEM
5429	QPZRTVFX	QSYS	1998/12/08	09: 35: 43	*OWNER	77824	*SYSTEM
3548	QPZSYNC	QSYS	1998/12/08	12: 27: 47	*OWNER	77824	*SYSTEM
0373	QPOLFFDC	QSYS	1998/12/14	02: 41: 26	*USER	40960	*I NHERI T
5737	QPOLFLOP	QSYS	1998/12/14	03: 16: 20	*OWNER	172032	*SYSTEM
6289	QPOZPCP2	QSYS	1998/12/14	02: 55: 46	*USER	36864	*SYSTEM
5738	QQQCSDBM	QSYS	1999/04/06	12: 42: 08	*USER	40960	*SYSTEM
6140	QQQDBMWL	QSYS	1999/08/11	20: 41: 04	*USER	69632	*SYSTEM
5739	QQQDSDBM	QSYS	1999/04/06	12: 42: 10	*USER	86016	*SYSTEM
5740	QQQESDBM	QSYS	1998/12/14	02: 42: 29	*USER	45056	*SYSTEM
3405	QQQGET	QSYS	2000/02/11	14: 06: 48	*USER	258048	*SYSTEM
5363	QQQORY	QSYS	1999/08/30	07: 59: 08	*USER	20480	*SYSTEM
6176	QQQOSDBM	QSYS	1998/12/14	02: 43: 04	*USER	40960	*SYSTEM
6161	QQQSSDBM	QSYS	1998/12/14	02: 43: 11	*USER	45056	*SYSTEM

Nbr	Name	Li b	Date	Time	Authori ty	Si ze	State
5745	QQQTEMP1	QSYS	1999/05/27	13: 19: 43	*OWNER	86016	*SYSTEM
5746	QQQTEMP2	QSYS	1999/05/27	13: 19: 46	*OWNER	20480	*SYSTEM
6345	QQQUDFAC	QSYS	1999/08/24	19: 12: 30	*USER	36864	*USER
2897	QRCVDTAQ	QSYS	1999/04/28	12: 09: 29	*USER	53248	*SYSTEM
3758	QREXQ	QSYS	1998/12/08	14: 55: 11	*OWNER	20480	*SYSTEM
3759	QREXVAR	QSYS	1998/12/08	14: 55: 20	*USER	20480	*SYSTEM
3432	QREXX	QSYS	1998/12/08	14: 55: 30	*USER	20480	*SYSTEM
3165	QRFPTHU	QSYS	1998/12/08	12: 41: 22	*OWNER	65536	*SYSTEM
3336	QRFHLT	QSYS	1998/12/08	12: 41: 12	*USER	49152	*SYSTEM
5446	QRZCHGE	QSYS	1998/12/08	15: 04: 27	*USER	28672	*SYSTEM
5442	QRZCRTH	QSYS	1998/12/08	09: 51: 47	*USER	32768	*SYSTEM
5445	QRZDLTE	QSYS	1998/12/08	12: 48: 50	*USER	28672	*SYSTEM
5447	QRZDLTH	QSYS	1998/12/08	12: 49: 01	*USER	28672	*SYSTEM
4440	QRZMI G4	QSYS	1998/11/18	02: 26: 50	*OWNER	8192	*SYSTEM
5448	QRZRRCRCA	QSYS	1998/12/08	15: 11: 07	*USER	159744	*SYSTEM
5458	QRZRRSI	QSYS	1998/12/08	12: 52: 02	*USER	98304	*SYSTEM
5459	QRZRTVR	QSYS	1998/12/08	15: 11: 19	*USER	32768	*SYSTEM
5460	QRZSCHE	QSYS	1998/12/08	12: 52: 50	*USER	118784	*SYSTEM
6142	QSCBMKTE	QSYS	1998/12/14	22: 36: 52	*USER	45056	*SYSTEM
6143	QSCBPSP	QSYS	1998/12/14	22: 36: 53	*USER	36864	*SYSTEM
6144	QSCBSRVE	QSYS	1998/12/14	22: 36: 54	*USER	53248	*SYSTEM
2057	QSCCPY	QSYS	1998/12/08	12: 54: 27	*OWNER	49152	*SYSTEM
5851	QSECSBD1	QSYS	1999/02/04	14: 17: 24	*USER	61440	*USER
5852	QSECSBD3	QSYS	1998/12/12	03: 48: 54	*USER	36864	*USER
1032	QSI GET	QSYS	1998/12/08	15: 26: 00	*USER	49152	*SYSTEM
1033	QSI PUT	QSYS	1998/12/08	10: 02: 22	*USER	65536	*SYSTEM
0796	QSLGET	QSYS	1998/12/08	13: 04: 07	*USER	81920	*SYSTEM
0795	QSLPUT	QSYS	1998/12/08	13: 04: 17	*USER	86016	*SYSTEM
2898	QSNDDTAQ	QSYS	1999/04/28	12: 09: 07	*USER	40960	*SYSTEM
5218	QSOCI P03	QSYS	1998/12/08	13: 08: 47	*OWNER	36864	*SYSTEM
5219	QSOCI P04	QSYS	1998/12/08	13: 07: 08	*OWNER	45056	*SYSTEM
5148	QSOCWI FC	QSYS	1998/12/08	15: 29: 29	*OWNER	36864	*SYSTEM
5150	QSOCWRTE	QSYS	1998/12/08	13: 10: 14	*OWNER	36864	*SYSTEM
5435	QSPBOPNC	QSYS	1998/12/08	13: 08: 18	*USER	32768	*SYSTEM
0045	QSPBPDSK	QSYS	1998/12/08	10: 06: 03	*USER	24576	*SYSTEM
5434	QSPBSEPP	QSYS	1998/12/08	15: 32: 02	*USER	53248	*SYSTEM
5433	QSPCHGOQ	QSYS	1999/07/01	16: 12: 25	*USER	32768	*SYSTEM
5053	QSPCLOSP	QSYS	1998/12/18	16: 16: 57	*OWNER	32768	*SYSTEM
5054	QSPCRTSP	QSYS	1999/05/04	17: 02: 51	*OWNER	225280	*SYSTEM
3315	QSPEXTWI	QSYS	1998/12/08	15: 34: 55	*OWNER	32768	*SYSTEM
0112	QSPFEOD	QSYS	1998/12/22	13: 03: 28	*OWNER	106496	*SYSTEM
4237	QSPFI XUP	QSYS	1998/12/18	16: 18: 53	*OWNER	81920	*SYSTEM
3928	QSPGETF	QSYS	1998/12/08	13: 15: 20	*OWNER	53248	*SYSTEM
5055	QSPGETSP	QSYS	1999/01/13	11: 34: 37	*OWNER	81920	*SYSTEM
5058	QSPMOVJB	QSYS	1998/12/08	10: 09: 22	*OWNER	69632	*SYSTEM
5057	QSPMOVSP	QSYS	1998/12/08	15: 36: 58	*USER	36864	*SYSTEM
5052	QSPOPNSP	QSYS	1998/12/18	16: 19: 12	*USER	40960	*SYSTEM
3929	QSPPUTF	QSYS	1998/12/08	15: 39: 44	*OWNER	28672	*SYSTEM
5056	QSPPUTSP	QSYS	1999/01/13	11: 42: 58	*OWNER	81920	*SYSTEM
0194	QSPRDR	QSYS	1998/12/08	13: 19: 44	*USER	102400	*SYSTEM
5051	QSPRJOBQ	QSYS	1998/12/08	15: 41: 02	*OWNER	40960	*SYSTEM
5050	QSPROUTQ	QSYS	1999/07/15	12: 07: 57	*OWNER	49152	*SYSTEM
5364	QSPRWTRI	QSYS	1998/12/08	13: 19: 59	*OWNER	28672	*SYSTEM
2650	QSPSETWI	QSYS	1998/12/08	13: 20: 59	*OWNER	61440	*SYSTEM
5197	QSPSNDWM	QSYS	1998/12/08	15: 42: 52	*OWNER	36864	*SYSTEM
0195	QSPWTRM1	QSYS	1999/01/30	21: 14: 47	*OWNER	65536	*SYSTEM
5064	QSQCHK	QSYS	1999/06/19	15: 16: 37	*USER	69632	*SYSTEM
4667	QSQCCLSE	QSYS	2000/02/09	12: 58: 40	*USER	49152	*SYSTEM
4668	QSQCMI T	QSYS	1999/07/16	09: 37: 43	*USER	32768	*SYSTEM
4666	QSQCOPEN	QSYS	2000/02/24	11: 53: 42	*USER	102400	*SYSTEM
5365	QSQPRCED	QSYS	2000/03/13	16: 48: 02	*USER	143360	*SYSTEM
2643	QSQRROUTE	QSYS	2000/02/17	18: 24: 14	*USER	1228800	*SYSTEM
6177	QSRCRTMD	QSYS	1998/12/14	03: 04: 09	*USER	40960	*SYSTEM
6178	QSRDLTMD	QSYS	1998/12/14	03: 04: 15	*USER	40960	*SYSTEM
2938	QSRDUPER	QSYS	1998/12/08	10: 19: 26	*USER	40960	*SYSTEM
1244	QSRFEOD	QSYS	1998/12/08	15: 54: 52	*USER	24576	*SYSTEM
1246	QSRGET	QSYS	1998/12/08	10: 19: 02	*USER	24576	*SYSTEM
5366	QSRLSAVF	QSYS	1999/01/13	11: 43: 08	*USER	57344	*SYSTEM
1248	QSRPUT	QSYS	1998/12/08	15: 56: 12	*USER	24576	*SYSTEM
6179	QSRRTVMD	QSYS	1998/12/14	03: 04: 20	*USER	40960	*SYSTEM
3249	QSRSAGO	QSYS	1999/01/13	11: 43: 32	*USER	106496	*SYSTEM
3880	QSRTGTRS	QSYS	1998/12/08	10: 22: 04	*OWNER	24576	*SYSTEM

Nbr	Name	Li b	Date	Time	Authori ty	Si ze	State
2679	QSUWHPCP	QSYS	1998/12/08	10: 23: 05	*USER	24576	*SYSTEM
4100	QSFTRPB	QSYS	1998/12/08	10: 31: 08	*USER	32768	*SYSTEM
4192	QSPRTPB	QSYS	1998/12/08	10: 30: 34	*USER	20480	*SYSTEM
3523	QSRMTR2	QSYS	1998/12/08	13: 46: 44	*USER	24576	*SYSTEM
5726	QSYADDUC	QSYS	1998/12/14	03: 20: 30	*USER	36864	*INHERIT
5727	QSYADDVC	QSYS	1998/12/14	03: 20: 35	*USER	36864	*INHERIT
5704	QSYADVLE	QSYS	1998/12/14	03: 01: 06	*USER	40960	*SYSTEM
6162	QSYCHFUI	QSYS	1998/12/14	03: 10: 52	*USER	40960	*INHERIT
5695	QSYCHGDS	QSYS	1998/12/08	16: 17: 54	*OWNER	45056	*SYSTEM
5693	QSYCHGID	QSYS	1999/01/13	13: 42: 44	*USER	36864	*SYSTEM
4865	QSYCHGPR	QSYS	1998/12/08	16: 17: 55	*OWNER	28672	*SYSTEM
4866	QSYCHGPW	QSYS	1999/08/25	10: 12: 28	*OWNER	61440	*SYSTEM
5728	QSYCHKVC	QSYS	1998/12/14	03: 20: 45	*USER	36864	*INHERIT
5705	QSYCHVLE	QSYS	1998/12/14	03: 01: 11	*USER	40960	*SYSTEM
6163	QSYCKUFU	QSYS	1998/12/14	03: 10: 57	*USER	36864	*INHERIT
4867	QSYCUSRA	QSYS	1998/12/08	10: 32: 18	*OWNER	57344	*SYSTEM
5078	QSYCUSRS	QSYS	1998/12/08	16: 18: 36	*OWNER	36864	*SYSTEM
4868	QSYCVTA	QSYS	1998/12/08	16: 18: 04	*USER	28672	*SYSTEM
6184	QSYDRGAP	QSYS	1998/12/14	03: 11: 02	*OWNER	36864	*INHERIT
6164	QSYDRGFN	QSYS	1998/12/14	03: 11: 07	*USER	36864	*INHERIT
5706	QSYFDVLE	QSYS	1998/12/14	03: 01: 16	*USER	40960	*SYSTEM
5729	QSYFNDCU	QSYS	1998/12/14	03: 21: 00	*USER	36864	*INHERIT
4869	QSYGETPH	QSYS	1999/01/13	13: 43: 39	*OWNER	40960	*SYSTEM
4871	QSYLATLO	QSYS	1999/01/13	11: 43: 49	*OWNER	86016	*SYSTEM
4870	QSYLAUTU	QSYS	1999/01/13	11: 43: 58	*USER	57344	*SYSTEM
4873	QSYLOBJA	QSYS	1999/01/13	11: 44: 10	*USER	77824	*SYSTEM
4872	QSYLOBJP	QSYS	1999/01/13	11: 44: 20	*OWNER	61440	*SYSTEM
5730	QSYLSTUC	QSYS	1998/12/14	03: 21: 05	*USER	36864	*INHERIT
5731	QSYLSTVC	QSYS	1998/12/14	03: 21: 10	*USER	36864	*INHERIT
4874	QSYLUSRA	QSYS	1999/01/13	11: 45: 19	*OWNER	77824	*SYSTEM
5732	QSYPARSC	QSYS	1998/12/14	03: 21: 15	*USER	36864	*INHERIT
5855	QSYPREAD	QSYS	1998/12/08	16: 21: 14	*USER	20480	*SYSTEM
5685	QSYRAUTU	QSYS	1998/12/09	04: 29: 37	*USER	36864	*SYSTEM
6185	QSYRGAP	QSYS	1998/12/14	03: 11: 12	*OWNER	36864	*INHERIT
6165	QSYRGFN	QSYS	1998/12/14	03: 11: 17	*USER	36864	*INHERIT
4875	QSYRLSPH	QSYS	1998/12/08	13: 53: 47	*OWNER	28672	*SYSTEM
5707	QSYRMVLE	QSYS	1998/12/14	03: 01: 21	*USER	36864	*SYSTEM
6195	QSYRMVPT	QSYS	1998/12/14	03: 17: 26	*USER	40960	*SYSTEM
5733	QSYRMVUC	QSYS	1998/12/14	03: 21: 19	*USER	36864	*INHERIT
5734	QSYRMVVC	QSYS	1998/12/14	03: 21: 24	*USER	36864	*INHERIT
6150	QSYRTCHI	QSYS	1998/12/08	13: 55: 20	*OWNER	32768	*SYSTEM
6166	QSYRTFUI	QSYS	1998/12/14	03: 11: 22	*USER	36864	*INHERIT
6167	QSYRTFUI	QSYS	1998/12/14	03: 11: 26	*USER	36864	*INHERIT
6168	QSYRTVFI	QSYS	1998/12/14	03: 11: 31	*USER	36864	*INHERIT
5701	QSYRTVSE	QSYS	1999/04/08	00: 52: 10	*USER	36864	*SYSTEM
5694	QSYRTVUA	QSYS	1998/12/14	03: 21: 34	*USER	53248	*SYSTEM
5723	QSYRUPWD	QSYS	1998/12/09	04: 29: 39	*USER	32768	*SYSTEM
4876	QSYRUSRA	QSYS	1998/12/08	10: 36: 46	*OWNER	57344	*SYSTEM
4877	QSYRUSRI	QSYS	1998/12/09	04: 29: 42	*USER	53248	*SYSTEM
5724	QSYSUPWD	QSYS	1999/08/25	10: 12: 15	*USER	40960	*SYSTEM
5686	QSZCHKTG	QSYS	1998/12/08	13: 56: 28	*USER	32768	*INHERIT
5367	QSZCRTPD	QSYS	1998/12/08	16: 24: 32	*USER	61440	*SYSTEM
5368	QSZCRTPL	QSYS	1998/12/08	13: 57: 09	*USER	69632	*SYSTEM
5369	QSZDLTPD	QSYS	1998/12/08	13: 57: 36	*USER	32768	*SYSTEM
5370	QSZDLTPL	QSYS	1998/12/08	10: 37: 40	*USER	32768	*SYSTEM
5371	QSZPKGPO	QSYS	1998/12/08	13: 57: 59	*USER	32768	*SYSTEM
1929	QSZRECOV	QSYS	1999/01/29	23: 55: 04	*OWNER	45056	*SYSTEM
5372	QSZRTVPR	QSYS	1998/12/08	10: 38: 24	*USER	73728	*SYSTEM
5373	QSZSLTPR	QSYS	1998/12/08	16: 25: 28	*OWNER	65536	*SYSTEM
6182	QSZSPTPR	QSYS	1999/01/29	23: 54: 57	*USER	77824	*SYSTEM
6169	QTACJMA	QSYS	1999/04/24	10: 11: 52	*USER	53248	*SYSTEM
6186	QTACTL DV	QSYS	1998/12/08	14: 02: 46	*USER	57344	*SYSTEM
5696	QTADMPDV	QSYS	1998/12/09	04: 30: 13	*USER	225280	*SYSTEM
0092	QTAFEOD	QSYS	1999/04/24	10: 09: 47	*USER	28672	*SYSTEM
0787	QTAFEOV	QSYS	1999/04/24	10: 09: 49	*USER	24576	*SYSTEM
0058	QTAGET	QSYS	1998/12/09	03: 06: 16	*USER	40960	*SYSTEM
5675	QTALCTG	QSYS	1998/11/18	02: 52: 54	*USER	8192	*SYSTEM
5676	QTAPI PE	QSYS	1998/11/18	02: 53: 09	*USER	8192	*SYSTEM
0059	QTAPUT	QSYS	1999/04/24	10: 10: 49	*USER	32768	*SYSTEM
5677	QTARDCAP	QSYS	1998/12/08	16: 31: 08	*USER	61440	*SYSTEM
5697	QTARDI NF	QSYS	1998/12/09	04: 31: 46	*USER	45056	*SYSTEM
6170	QTARJMA	QSYS	1998/12/08	14: 06: 34	*USER	49152	*SYSTEM

Nbr	Name	Li b	Date	Time	Authori ty	Si ze	State
2823	QTBXLATE	QSYS	1998/12/08	14: 09: 04	*USER	61440	*SYSTEM
6081	QTEBRKWK	QSYS	1998/12/14	02: 54: 01	*USER	36864	*USER
6082	QTEDFI	QSYS	1998/12/14	02: 54: 03	*USER	45056	*I NHERI T
6087	QTEEVWLK	QSYS	1998/12/14	02: 54: 09	*USER	36864	*USER
6089	QTEMODN	QSYS	1998/12/14	02: 54: 13	*USER	61440	*SYSTEM
6090	QTEPGMWK	QSYS	1998/12/14	02: 54: 15	*USER	36864	*USER
6091	QTEPLSWK	QSYS	1998/12/14	02: 54: 16	*USER	36864	*USER
5374	QTERTPVP	QSYS	1998/12/08	14: 12: 28	*USER	49152	*SYSTEM
6092	QTESESSH	QSYS	1999/02/19	19: 30: 28	*USER	53248	*USER
6093	QTESTOPH	QSYS	1999/02/19	19: 30: 30	*USER	45056	*USER
6099	QTEHDWK	QSYS	1998/12/14	02: 54: 33	*USER	36864	*USER
6100	QTEWCHWK	QSYS	1998/12/14	02: 54: 40	*USER	36864	*USER
4881	QTNADDCR	QSYS	1998/12/08	10: 48: 01	*USER	57344	*SYSTEM
5375	QTNCHGCO	QSYS	1998/12/08	16: 37: 05	*USER	36864	*SYSTEM
1103	QTNCMT	QSYS	1999/07/29	19: 09: 03	*OWNER	86016	*SYSTEM
5376	QTNBRBDQ	QSYS	1998/12/08	16: 38: 56	*USER	36864	*SYSTEM
4880	QTNRCMTI	QSYS	1998/12/08	14: 17: 56	*USER	32768	*SYSTEM
4882	QTNRMVCR	QSYS	1998/12/08	16: 38: 43	*USER	32768	*SYSTEM
1102	QTNROLLB	QSYS	1999/08/09	18: 52: 31	*OWNER	139264	*SYSTEM
4524	QTOAABRT	QSYS	1998/12/08	10: 52: 09	*OWNER	24576	*SYSTEM
4525	QTOAAPI D	QSYS	1998/12/08	16: 39: 42	*OWNER	20480	*SYSTEM
4522	QTOAAPI U	QSYS	1998/12/08	16: 40: 01	*OWNER	24576	*SYSTEM
3964	QTOACLS	QSYS	1998/12/09	03: 19: 03	*OWNER	24576	*SYSTEM
4514	QTOAEND	QSYS	1998/12/09	03: 19: 11	*OWNER	20480	*SYSTEM
4527	QTOAGHBN	QSYS	1998/12/08	14: 21: 22	*OWNER	20480	*SYSTEM
4515	QTOAGTI D	QSYS	1998/12/08	10: 51: 54	*OWNER	24576	*SYSTEM
4528	QTOAI SLA	QSYS	1998/12/08	16: 41: 20	*OWNER	24576	*SYSTEM
4530	QTOAI SUP	QSYS	1998/12/08	16: 41: 11	*OWNER	24576	*SYSTEM
3956	QTOAOPN	QSYS	1998/12/08	14: 20: 56	*OWNER	45056	*SYSTEM
3953	QTOAPI NG	QSYS	1998/12/08	14: 21: 14	*OWNER	32768	*SYSTEM
4529	QTOAQRYD	QSYS	1998/12/08	10: 51: 34	*OWNER	24576	*SYSTEM
4526	QTOARCV	QSYS	1998/12/09	04: 31: 52	*OWNER	24576	*SYSTEM
3957	QTOASND	QSYS	1998/12/09	21: 07: 35	*OWNER	24576	*SYSTEM
4516	QTOASTRM	QSYS	1998/12/08	14: 21: 14	*OWNER	40960	*SYSTEM
3954	QTOASTT	QSYS	1998/12/09	03: 12: 28	*OWNER	24576	*SYSTEM
4517	QTOATI ME	QSYS	1998/12/08	10: 52: 24	*OWNER	28672	*SYSTEM
4520	QTOAUCLS	QSYS	1998/12/09	03: 19: 16	*OWNER	24576	*SYSTEM
4518	QTOAUOPN	QSYS	1998/12/08	16: 41: 47	*OWNER	28672	*SYSTEM
4521	QTOAURCV	QSYS	1998/12/09	03: 19: 46	*OWNER	24576	*SYSTEM
4519	QTOAUSND	QSYS	1998/12/09	04: 31: 54	*OWNER	24576	*SYSTEM
3955	QTOAUSTT	QSYS	1998/12/09	04: 31: 54	*OWNER	24576	*SYSTEM
6012	QTOCAUTO	QSYS	1999/05/14	13: 55: 00	*OWNER	32768	*SYSTEM
6015	QTOCI FCU	QSYS	1999/10/05	14: 41: 43	*OWNER	36864	*SYSTEM
6187	QTOCLPPJ	QSYS	1998/12/14	03: 17: 37	*USER	139264	*SYSTEM
6349	QTOCL2TP	QSYS	1999/07/29	15: 49: 16	*USER	843776	*USER
6017	QTOCNETC	QSYS	1999/05/14	13: 54: 56	*OWNER	73728	*SYSTEM
6066	QTOCPORU	QSYS	1998/12/08	16: 44: 00	*OWNER	32768	*SYSTEM
6021	QTOCPPMU	QSYS	1999/10/05	14: 42: 35	*OWNER	114688	*SYSTEM
6350	QTOCPPPCTL	QSYS	1999/06/24	21: 49: 29	*OWNER	196608	*SYSTEM
6022	QTOCPPPM	QSYS	1999/01/31	19: 55: 07	*OWNER	45056	*SYSTEM
6023	QTOCPPPU	QSYS	1999/07/21	06: 52: 16	*OWNER	69632	*SYSTEM
5862	QTOCPPSM	QSYS	1999/10/05	14: 42: 56	*OWNER	65536	*SYSTEM
6024	QTOCRTEU	QSYS	1998/12/08	16: 44: 59	*OWNER	36864	*SYSTEM
5898	QTOCRTTC	QSYS	1998/12/08	16: 45: 49	*USER	40960	*SYSTEM
6055	QTOddb2D	QSYS	1998/12/14	03: 11: 36	*OWNER	106496	*SYSTEM
6254	QTOGI NTD	QSYS	1999/01/05	15: 40: 04	*USER	139264	*SYSTEM
6255	QTOGI NTE	QSYS	1998/12/14	02: 46: 46	*USER	49152	*USER
4578	QTOSMAI N	QSYS	1998/12/14	02: 58: 56	*OWNER	40960	*SYSTEM
4583	QTOSRCVR	QSYS	1998/12/14	02: 59: 01	*OWNER	45056	*SYSTEM
5424	QTOQVRT	QSYS	1999/01/15	17: 32: 10	*OWNER	49152	*SYSTEM
5425	QTOGCCN	QSYS	1998/12/08	16: 50: 21	*OWNER	24576	*SYSTEM
5444	QTOGCTL	QSYS	1998/12/08	14: 32: 58	*OWNER	45056	*SYSTEM
5426	QTOGESP	QSYS	1998/12/08	14: 33: 09	*OWNER	24576	*SYSTEM
5427	QTOGRDC	QSYS	1998/12/08	16: 49: 58	*OWNER	24576	*SYSTEM
5865	QTOI SOPM	QSYS	1998/12/14	02: 32: 40	*USER	40960	*SYSTEM
5698	QTOQRCSC	QSYS	1998/12/08	14: 34: 03	*USER	20480	*I NHERI T
5231	QTORECOV	QSYS	1998/12/08	10: 59: 36	*USER	24576	*SYSTEM
5428	QTOQSCSP	QSYS	1998/12/08	16: 50: 47	*OWNER	24576	*SYSTEM
5443	QTOQMXC	QSYS	1998/12/08	14: 34: 11	*USER	20480	*I NHERI T
5700	QTOQMXC2	QSYS	1998/12/08	14: 34: 09	*USER	20480	*I NHERI T
1631	QTSFEOD	QSYS	1998/12/08	16: 52: 12	*USER	77824	*SYSTEM
1632	QTSGET	QSYS	1998/12/09	03: 23: 11	*USER	61440	*SYSTEM

Nbr	Name	Li b	Date	Time	Authori ty	Si ze	State
1633	QTSGETM	QSYS	1998/12/08	14: 35: 55	*USER	53248	*SYSTEM
1685	QTSGETSQ	QSYS	1998/12/08	14: 35: 20	*USER	20480	*SYSTEM
5751	QTSGETK	QSYS	1998/12/22	01: 11: 13	*USER	245760	*SYSTEM
5752	QTSGETM	QSYS	1999/10/18	10: 46: 21	*USER	180224	*SYSTEM
5753	QTSGETS	QSYS	1998/12/22	01: 12: 55	*USER	262144	*SYSTEM
5756	QTSPPUT	QSYS	1998/12/22	01: 13: 33	*USER	94208	*SYSTEM
5757	QTSPPUTM	QSYS	1998/12/22	01: 13: 57	*USER	90112	*SYSTEM
5758	QTSPU DR	QSYS	1998/12/22	01: 08: 40	*USER	139264	*SYSTEM
1623	QTSPUT	QSYS	1998/12/08	16: 54: 08	*USER	69632	*SYSTEM
2284	QTSPUTDR	QSYS	1998/12/08	16: 54: 01	*USER	49152	*SYSTEM
1624	QTSPUTM	QSYS	1998/12/08	14: 37: 25	*USER	57344	*SYSTEM
1621	QTSUDR	QSYS	1998/12/08	11: 07: 47	*USER	45056	*SYSTEM
5377	QTVCLQVT	QSYS	1998/12/09	03: 26: 03	*USER	28672	*SYSTEM
6160	QTVDMGR	QSYS	1998/12/08	14: 43: 06	*USER	20480	*SYSTEM
6231	QTVDMI EX	QSYS	1998/12/08	14: 43: 17	*USER	20480	*SYSTEM
6232	QTVENDTN	QSYS	1998/12/08	16: 57: 06	*USER	20480	*SYSTEM
5378	QTVOPNVT	QSYS	1998/12/09	04: 35: 00	*USER	53248	*SYSTEM
5379	QTVRDVT	QSYS	1998/12/09	03: 19: 07	*USER	32768	*SYSTEM
5380	QTVSNDRO	QSYS	1998/12/09	04: 35: 35	*USER	32768	*SYSTEM
6234	QTVSTRTN	QSYS	1998/12/08	16: 58: 26	*USER	24576	*SYSTEM
6235	QTVTELN	QSYS	1998/12/14	02: 48: 48	*USER	57344	*SYSTEM
5381	QTVWRTVT	QSYS	1998/12/09	03: 29: 48	*USER	36864	*SYSTEM
5382	QTWAI DSP	QSYS	1999/04/21	14: 15: 41	*OWNER	49152	*SYSTEM
5384	QTVCHKSP	QSYS	1999/04/21	14: 15: 32	*OWNER	49152	*SYSTEM
4843	QUHDSHP	QSYS	1998/12/08	14: 49: 58	*USER	32768	*SYSTEM
4844	QUHPRTH	QSYS	1998/12/08	17: 03: 41	*USER	24576	*SYSTEM
4845	QUI ADDLE	QSYS	1998/12/08	17: 03: 47	*USER	28672	*SYSTEM
4846	QUI ADDLM	QSYS	1998/12/08	14: 50: 55	*USER	28672	*SYSTEM
4847	QUI ADDPA	QSYS	1998/12/08	14: 51: 31	*USER	28672	*SYSTEM
4848	QUI ADDPW	QSYS	1998/12/08	11: 14: 52	*USER	28672	*SYSTEM
4849	QUI CLOA	QSYS	1998/12/08	14: 52: 12	*USER	28672	*SYSTEM
4850	QUI DLT L	QSYS	1998/12/08	17: 05: 07	*USER	24576	*SYSTEM
4851	QUI DSPP	QSYS	1998/12/08	14: 52: 15	*USER	28672	*SYSTEM
4852	QUI GETLE	QSYS	1998/12/08	17: 05: 31	*USER	28672	*SYSTEM
4853	QUI GETLM	QSYS	1998/12/08	14: 53: 17	*USER	28672	*SYSTEM
4854	QUI GETV	QSYS	1998/12/08	14: 53: 11	*USER	24576	*SYSTEM
5674	QUI LNGTX	QSYS	1998/12/08	11: 15: 52	*USER	32768	*SYSTEM
4855	QUI OPNDA	QSYS	1998/12/08	11: 16: 42	*USER	28672	*SYSTEM
4856	QUI OPNPA	QSYS	1998/12/08	17: 06: 32	*USER	28672	*SYSTEM
4857	QUI PRTP	QSYS	1998/12/08	11: 17: 51	*USER	24576	*SYSTEM
5385	QUI PUTV	QSYS	1998/12/08	17: 07: 06	*USER	24576	*SYSTEM
4858	QUI RMVLE	QSYS	1998/12/08	14: 56: 19	*USER	24576	*SYSTEM
4859	QUI RMVPA	QSYS	1998/12/08	14: 56: 06	*USER	28672	*SYSTEM
4860	QUI RMVPW	QSYS	1998/12/08	11: 18: 32	*USER	24576	*SYSTEM
5386	QUI RTVLA	QSYS	1998/12/08	17: 08: 01	*USER	24576	*SYSTEM
4861	QUI SETLA	QSYS	1998/12/08	14: 57: 23	*USER	24576	*SYSTEM
4862	QUI SETSC	QSYS	1998/12/08	11: 17: 52	*USER	24576	*SYSTEM
4863	QUI UPDLE	QSYS	1998/12/08	17: 08: 41	*USER	28672	*SYSTEM
6236	QUPMENU	QSYS	1998/11/18	03: 42: 37	*USER	8192	*USER
6237	QUPVLI C	QSYS	1998/11/18	03: 43: 05	*USER	16384	*USER
4917	QUSADDEP	QSYS	1998/12/14	03: 05: 13	*USER	36864	*INHERI T
5387	QUSADDUI	QSYS	1999/01/13	11: 44: 46	*USER	49152	*SYSTEM
4928	QUSCHGPA	QSYS	1998/12/08	14: 58: 59	*USER	40960	*SYSTEM
5388	QUSCHGUS	QSYS	1999/01/13	11: 44: 39	*USER	53248	*SYSTEM
1852	QUSCMDLN	QSYS	1998/12/08	17: 10: 00	*USER	24576	*SYSTEM
5389	QUSCRTUI	QSYS	1998/12/08	17: 10: 33	*USER	53248	*SYSTEM
5390	QUSCRTUQ	QSYS	1998/12/08	14: 59: 29	*USER	53248	*SYSTEM
5391	QUSCRTUS	QSYS	1999/03/24	16: 12: 15	*USER	53248	*SYSTEM
5392	QUSCUSAT	QSYS	1999/01/13	11: 44: 31	*USER	53248	*SYSTEM
5393	QUSDLTUI	QSYS	1998/12/08	17: 10: 20	*USER	28672	*SYSTEM
5394	QUSDLTUQ	QSYS	1998/12/08	17: 10: 11	*USER	28672	*SYSTEM
5395	QUSDLTUS	QSYS	1998/12/08	14: 59: 20	*USER	28672	*SYSTEM
4916	QUSDRGPT	QSYS	1998/12/14	03: 05: 18	*USER	36864	*INHERI T
5404	QUSLFLD	QSYS	1999/01/30	11: 39: 55	*OWNER	77824	*SYSTEM
4929	QUSLJOB	QSYS	1999/01/13	11: 42: 32	*OWNER	147456	*SYSTEM
5093	QUSLMBR	QSYS	1999/03/23	22: 00: 24	*OWNER	106496	*SYSTEM
4950	QUSLOBJ	QSYS	1999/01/13	11: 39: 43	*OWNER	81920	*SYSTEM
5405	QUSLRCD	QSYS	1999/01/13	11: 39: 54	*OWNER	73728	*SYSTEM
5049	QUSLSPL	QSYS	1999/05/04	17: 01: 38	*OWNER	110592	*SYSTEM
5396	QUSPTRUS	QSYS	1998/12/08	17: 10: 39	*USER	36864	*SYSTEM
4636	QUSRGFI N	QSYS	1999/02/19	13: 24: 40	*USER	94208	*SYSTEM
4915	QUSRGPT	QSYS	1998/12/14	03: 05: 33	*USER	36864	*INHERI T

Nbr	Name	Li b	Date	Time	Authori ty	Si ze	State
4898	QUSRJOBI	QSYS	1998/12/09	03: 31: 27	*OWNER	81920	*SYSTEM
5094	QUSRMBRD	QSYS	1999/02/04	14: 25: 31	*USER	98304	*SYSTEM
4918	QUSRMVEP	QSYS	1998/12/14	03: 05: 38	*USER	36864	*I NHERI T
5397	QUSRMVUI	QSYS	1999/09/29	12: 09: 37	*USER	57344	*SYSTEM
4951	QUSROBJD	QSYS	1998/12/08	11: 20: 40	*USER	36864	*SYSTEM
5048	QUSRSPLA	QSYS	1998/12/08	17: 11: 28	*USER	65536	*SYSTEM
4914	QUSRTVEI	QSYS	1998/12/14	03: 05: 44	*USER	36864	*I NHERI T
5398	QUSRTVUI	QSYS	1999/09/29	12: 09: 27	*USER	57344	*SYSTEM
5399	QUSRTVUS	QSYS	1999/01/13	11: 40: 41	*USER	53248	*SYSTEM
5400	QUSRUI AT	QSYS	1999/01/13	11: 40: 47	*USER	49152	*SYSTEM
5401	QUSRUSAT	QSYS	1999/03/22	09: 12: 38	*USER	53248	*SYSTEM
5402	QVTRMSTG	QSYS	1998/12/08	17: 14: 43	*OWNER	28672	*SYSTEM
4931	QWCCCJOB	QSYS	1998/12/08	15: 08: 22	*USER	32768	*SYSTEM
4932	QWCCHGTN	QSYS	1998/12/08	15: 10: 09	*USER	32768	*SYSTEM
0086	QWCCRCRC	QSYS	1998/12/08	17: 17: 26	*USER	24576	*SYSTEM
4933	QWCCVTDI	QSYS	1998/12/08	15: 11: 56	*USER	40960	*SYSTEM
4934	QWCLASBS	QSYS	1999/01/13	11: 41: 03	*OWNER	53248	*SYSTEM
4935	QWCLOBJL	QSYS	1999/01/13	11: 41: 14	*OWNER	73728	*SYSTEM
4936	QWCLSCDE	QSYS	1999/01/13	11: 41: 27	*OWNER	81920	*SYSTEM
6180	QWCRCLSI	QSYS	1998/12/08	17: 21: 34	*USER	28672	*SYSTEM
4937	QWCRDTAA	QSYS	1998/12/08	17: 21: 32	*USER	32768	*SYSTEM
5708	QWCRI PLA	QSYS	1998/12/08	15: 15: 39	*USER	28672	*SYSTEM
5692	QWCRIJBT	QSYS	1998/12/08	15: 17: 51	*OWNER	40960	*SYSTEM
4938	QWCRNETA	QSYS	1998/12/08	11: 30: 16	*USER	45056	*SYSTEM
4939	QWCRSSTS	QSYS	1998/12/08	17: 22: 32	*USER	73728	*SYSTEM
4940	QWCRSVAL	QSYS	1998/12/08	17: 21: 51	*USER	73728	*SYSTEM
5709	QWCRTVCA	QSYS	1998/12/14	03: 21: 45	*USER	98304	*SYSTEM
5980	QWCSCLXR	QSYS	1998/12/08	17: 22: 05	*OWNER	24576	*SYSTEM
0081	QWCSVRDR	QSYS	1998/12/08	15: 17: 29	*USER	20480	*SYSTEM
5805	QWCSVRD2	QSYS	1998/12/08	15: 17: 45	*USER	20480	*SYSTEM
0082	QWCSVRTR	QSYS	1998/12/08	17: 22: 55	*USER	20480	*SYSTEM
6181	QWDCSBSE	QSYS	1998/12/14	03: 22: 00	*USER	53248	*SYSTEM
5687	QWDLSEBSE	QSYS	1999/01/13	11: 41: 38	*USER	73728	*SYSTEM
4941	QWDLSEJBO	QSYS	1999/01/13	11: 41: 48	*OWNER	65536	*SYSTEM
4942	QWDRJOBDB	QSYS	1998/12/08	11: 33: 29	*USER	36864	*SYSTEM
4943	QWDRSBSDB	QSYS	1998/12/08	17: 25: 30	*OWNER	36864	*SYSTEM
4026	QWPAPTFL	QSYS	1999/09/01	13: 17: 11	*USER	217088	*SYSTEM
4025	QWPAPUT	QSYS	1999/09/01	13: 15: 08	*USER	94208	*SYSTEM
1885	QWPASPR	QSYS	1998/12/08	17: 26: 28	*USER	28672	*SYSTEM
0107	QWPFEOD	QSYS	1998/12/08	11: 34: 46	*USER	40960	*SYSTEM
1232	QWPGRAPH	QSYS	1998/12/08	17: 26: 48	*USER	28672	*SYSTEM
1540	QWPIPTFL	QSYS	1999/09/01	13: 15: 41	*USER	126976	*SYSTEM
1610	QWPIPUT	QSYS	1998/12/08	11: 37: 16	*USER	86016	*SYSTEM
1819	QWPLNPR	QSYS	1999/05/07	10: 05: 56	*USER	73728	*SYSTEM
6122	QWPLPTFL	QSYS	1999/09/01	13: 16: 04	*USER	94208	*SYSTEM
1954	QWPPTFLD	QSYS	1999/09/01	13: 14: 46	*USER	122880	*SYSTEM
1953	QWPPUT	QSYS	1998/12/08	17: 29: 37	*USER	106496	*SYSTEM
4027	QWPXPUT	QSYS	1998/11/18	02: 31: 49	*USER	8192	*SYSTEM
5432	QWPZHPTR	QSYS	1998/12/14	03: 05: 49	*USER	36864	*I NHERI T
5406	QWPZTAFF	QSYS	1998/12/08	17: 28: 31	*USER	20480	*SYSTEM
0052	QWSGET	QSYS	2000/07/22	14: 56: 56	*USER	397312	*SYSTEM
1544	QWSPUDDS	QSYS	2000/04/12	17: 14: 41	*USER	94208	*SYSTEM
0022	QWSPUT	QSYS	2000/07/21	14: 46: 14	*USER	544768	*SYSTEM
1604	QWSORYWS	QSYS	1998/12/08	17: 30: 52	*USER	24576	*SYSTEM
5457	QWSRTVOI	QSYS	2000/02/23	11: 55: 45	*USER	53248	*SYSTEM
1158	QWSSETWS	QSYS	1998/12/08	15: 28: 21	*USER	24576	*SYSTEM
3762	QWSYSRQ	QSYS	1998/12/08	17: 31: 17	*USER	16384	*SYSTEM
5594	QWTATNAD	QSYS	1998/12/08	15: 30: 02	*OWNER	20480	*SYSTEM
5702	QWTCHGJB	QSYS	1999/05/14	14: 53: 29	*USER	270336	*SYSTEM
5407	QWTCTLTR	QSYS	1998/12/08	17: 34: 01	*USER	28672	*SYSTEM
5408	QWTDMPFR	QSYS	1998/12/08	17: 34: 28	*USER	28672	*SYSTEM
5409	QWTDMPFL	QSYS	1998/12/09	03: 28: 37	*USER	65536	*SYSTEM
5735	QWTDJTBS	QSYS	1998/12/08	15: 31: 40	*USER	32768	*SYSTEM
6354	QWTGETRQ	QSYS	1998/12/08	11: 41: 12	*USER	32768	*SYSTEM
6355	QWTPUTRQ	QSYS	1998/12/08	17: 39: 14	*USER	36864	*SYSTEM
5688	QWTRVPX	QSYS	1998/12/08	17: 39: 33	*USER	28672	*SYSTEM
5410	QWTSETLF	QSYS	1998/12/09	04: 46: 35	*USER	28672	*SYSTEM
4944	QWTSETP	QSYS	1998/12/08	11: 48: 41	*OWNER	40960	*SYSTEM
5689	QWTSETPX	QSYS	1998/12/08	17: 40: 50	*USER	28672	*SYSTEM
5411	QWTSETTR	QSYS	1998/12/08	17: 40: 53	*USER	28672	*SYSTEM
6171	QWTSJUI D	QSYS	1998/12/14	03: 22: 10	*USER	40960	*SYSTEM
4417	QWVPDAGE	QSYS	1998/12/08	15: 40: 17	*USER	16384	*I NHERI T

Nbr	Name	Li b	Date	Time	Authori ty	Si ze	State
6063	QXNAUPDF	QSYS	1999/09/01	10: 44: 19	*OWNER	77824	*SYSTEM
5868	QXOTRCVP	QSYS	1998/12/08	17: 41: 01	*OWNER	16384	*SYSTEM
6360	QYPSCDMC	QSYS	1999/01/26	15: 35: 01	*OWNER	176128	*SYSTEM
6189	QYPSCSCA	QSYS	1999/01/26	15: 35: 08	*OWNER	36864	*I NHERI T
6190	QYPSCYCC	QSYS	1999/01/26	15: 35: 10	*OWNER	36864	*I NHERI T
6191	QYPSENDC	QSYS	1999/01/26	15: 35: 21	*OWNER	36864	*I NHERI T
6172	QYPSENDS	QSYS	1999/01/26	15: 35: 23	*OWNER	49152	*SYSTEM
6192	QYPSRSCA	QSYS	1999/01/26	15: 35: 30	*OWNER	36864	*I NHERI T
6193	QYPSSTRC	QSYS	1999/01/26	15: 35: 40	*OWNER	36864	*I NHERI T
6173	QYPSSTRS	QSYS	1999/01/26	15: 35: 43	*OWNER	45056	*SYSTEM
6194	QYPSSTRT	QSYS	1999/01/26	15: 35: 45	*OWNER	36864	*I NHERI T
6365	QYSMCSVE	QSYS	1999/01/18	12: 31: 18	*OWNER	53248	*SYSTEM
6366	QYSMPUT	QSYS	1999/04/07	22: 09: 44	*USER	61440	*SYSTEM
6367	QYSMRSVE	QSYS	1998/12/14	03: 22: 50	*USER	45056	*SYSTEM
6070	QYUNLANG	QSYS	1998/12/14	02: 49: 50	*USER	40960	*USER
5311	QYYCCLOS	QSYS	1998/12/08	11: 50: 41	*USER	69632	*SYSTEM
5318	QYYCFEOD	QSYS	1998/12/09	03: 36: 17	*USER	57344	*SYSTEM
1101	QYYCGETD	QSYS	1998/12/09	03: 36: 34	*USER	61440	*SYSTEM
0701	QYYCGETK	QSYS	1998/12/09	04: 49: 00	*USER	61440	*SYSTEM
5314	QYYCGETM	QSYS	1998/12/09	04: 49: 56	*USER	61440	*SYSTEM
5312	QYYCGETS	QSYS	1998/12/09	03: 28: 44	*USER	61440	*SYSTEM
5315	QYYCPUT	QSYS	1998/12/09	03: 37: 08	*USER	53248	*SYSTEM
5317	QYYCPUTD	QSYS	1998/12/09	03: 37: 07	*USER	53248	*SYSTEM
5316	QYYCPUTM	QSYS	1998/12/09	04: 51: 01	*USER	57344	*SYSTEM
5319	QYYCUDR	QSYS	1998/12/09	04: 51: 01	*USER	61440	*SYSTEM
2488	QY11TFAT	QSYS	1998/12/08	15: 43: 40	*USER	16384	*SYSTEM
1703	QY2FTML	QSYS	1998/12/09	03: 30: 00	*USER	151552	*SYSTEM
5059	QZCAADDC	QSYS	1998/12/14	03: 08: 21	*OWNER	36864	*USER
5061	QZCAREFC	QSYS	1998/12/14	03: 08: 31	*OWNER	36864	*USER
5060	QZCARMVC	QSYS	1998/12/14	03: 08: 36	*OWNER	36864	*USER
5112	QZCATHR	QSYS	1998/12/14	03: 08: 41	*OWNER	102400	*SYSTEM
5062	QZCAUPDC	QSYS	1998/12/14	03: 08: 46	*OWNER	36864	*USER
5030	QZDAGFS	QSYS	1998/11/18	04: 20: 44	*USER	8192	*SYSTEM
5452	QZDASNI D	QSYS	1998/12/08	11: 51: 04	*OWNER	32768	*SYSTEM
5449	QZDCRFSD	QSYS	1998/12/08	17: 45: 16	*OWNER	36864	*SYSTEM
5454	QZDLSTI D	QSYS	1999/01/13	11: 41: 57	*OWNER	61440	*SYSTEM
5451	QZDRDFSO	QSYS	1998/12/08	15: 48: 08	*OWNER	40960	*SYSTEM
5455	QZDRTVI D	QSYS	1998/12/08	15: 48: 31	*OWNER	36864	*SYSTEM
5453	QZDRVKI D	QSYS	1998/12/08	17: 46: 58	*OWNER	32768	*SYSTEM
5450	QZDWTFSD	QSYS	1998/12/08	17: 48: 17	*OWNER	40960	*SYSTEM
5710	QZLSADFS	QSYS	1999/04/13	12: 58: 12	*USER	94208	*SYSTEM
5711	QZLSADPS	QSYS	1999/04/13	12: 58: 16	*USER	86016	*SYSTEM
5712	QZLSCHFSD	QSYS	1999/04/13	12: 58: 20	*USER	102400	*SYSTEM
5713	QZLSCHPS	QSYS	1999/04/13	12: 58: 24	*USER	69632	*SYSTEM
5714	QZLSCHSG	QSYS	1999/04/13	12: 58: 28	*USER	86016	*SYSTEM
5715	QZLSCHSI	QSYS	1999/04/13	12: 58: 31	*USER	86016	*SYSTEM
5716	QZLSCHSN	QSYS	1999/04/13	12: 58: 34	*USER	81920	*SYSTEM
5717	QZLSENDS	QSYS	1999/04/13	12: 58: 38	*USER	53248	*SYSTEM
5718	QZLSENSS	QSYS	1999/04/13	12: 58: 41	*USER	57344	*SYSTEM
5719	QZLSLSTI	QSYS	1999/06/22	19: 54: 34	*USER	139264	*SYSTEM
5722	QZLSLST	QSYS	1999/04/13	12: 58: 48	*USER	69632	*SYSTEM
5720	QZLSRMS	QSYS	1999/04/13	12: 58: 51	*USER	65536	*SYSTEM
5721	QZLSSTRS	QSYS	1999/04/13	12: 58: 54	*USER	57344	*SYSTEM
5065	QZMFACHG	QSYS	1998/12/08	17: 49: 41	*OWNER	204800	*SYSTEM
5066	QZMFACRT	QSYS	1998/12/08	15: 51: 00	*OWNER	135168	*SYSTEM
5067	QZMFADDC	QSYS	1998/12/08	15: 50: 34	*OWNER	40960	*SYSTEM
5069	QZMFALOG	QSYS	1998/11/18	04: 25: 40	*USER	8192	*SYSTEM
5070	QZMFAPG1	QSYS	1998/11/18	02: 43: 13	*USER	8192	*SYSTEM
5430	QZMFAORY	QSYS	1998/12/08	17: 48: 49	*OWNER	32768	*SYSTEM
5071	QZMFARVS	QSYS	1998/12/08	17: 49: 17	*OWNER	32768	*SYSTEM
5072	QZMFARTV	QSYS	1998/12/08	15: 51: 29	*OWNER	49152	*SYSTEM
5073	QZMFASCR	QSYS	1998/12/08	15: 51: 06	*OWNER	32768	*SYSTEM
5074	QZMFASQC	QSYS	1998/12/08	11: 55: 31	*OWNER	32768	*SYSTEM
5075	QZMFCATR	QSYS	1998/11/18	04: 26: 13	*USER	8192	*SYSTEM
5076	QZMFDLTC	QSYS	1998/12/08	17: 50: 15	*OWNER	40960	*SYSTEM
5077	QZMFLSTC	QSYS	1999/01/13	11: 32: 44	*OWNER	65536	*SYSTEM
2987	QZMFXDI R	QSYS	1998/12/08	15: 52: 40	*OWNER	40960	*SYSTEM
5262	QZMFXP G1	QSYS	1998/12/08	17: 50: 51	*OWNER	20480	*SYSTEM
5249	QZMFXP G2	QSYS	1998/11/18	02: 46: 09	*USER	16384	*SYSTEM
5250	QZMFXP G3	QSYS	1998/11/18	02: 46: 28	*USER	8192	*SYSTEM
5251	QZMFXP G4	QSYS	1998/11/17	22: 14: 58	*USER	8192	*SYSTEM
5725	QZNFRTVE	QSYS	1998/12/14	03: 24: 46	*USER	53248	*SYSTEM



Nbr	Name	Li b	Date	Time	Authori ty	Si ze	State
4905	QZPAI JOB	QSYS	1998/12/08	15: 54: 03	*OWNER	53248	*SYSTEM
6329	QZRUEDTF	QSYS	2000/01/18	15: 33: 54	*USER	299008	*USER
6064	QZSPI PI S	QSYS	1998/12/08	15: 55: 28	*OWNER	16384	*SYSTEM
3741	QZSPMJOB	QSYS	1998/12/08	15: 57: 04	*OWNER	40960	*SYSTEM
4442	QZSPWI PA	QSYS	1998/12/08	12: 00: 15	*OWNER	32768	*SYSTEM
4439	QZSPWI PR	QSYS	1998/12/08	15: 58: 26	*OWNER	32768	*SYSTEM
4954	QZSPWLOC	QSYS	1998/12/08	17: 55: 03	*OWNER	32768	*SYSTEM
1878	SUBRA1	QSYS	1998/12/08	17: 55: 57	*USER	24576	*SYSTEM
1876	SUBRF1	QSYS	1998/11/18	02: 52: 50	*USER	20480	*USER
1877	SUBRF2	QSYS	1998/11/17	22: 19: 18	*USER	20480	*USER

## Appendix E

### SCV 10 Function Numbers

#### Supervisor Call (Vectored) Functions

When the **SCV** (Supervisor Call Vectored) instruction is executed with an operand of 10, General Purpose Register 10 holds the function number. It is a coincidence that the number 10 occurs twice in the above statement. Registers R3 and up are used for parameters of the call. Most complicated MI-instructions are implemented as SCV calls.

The following table contains information about each function as far as it is known. The module names and the addresses may change from release to release (I know of several such changes). The function number in R10 and the corresponding MI-instruction form a very stable pair. There are only a few cases where a function that was earlier executed through SCV 10 is now done differently. These cases are marked by using a fainter font face, and by having no dispatch address defined; an example is the EDIT MI-instruction. The 'B' column indicates by a Y(es) or N(o) if the instruction is blocked at security level 40 or higher.

R10	MI	Module	Description	B	Address V4R4M0
1	CVTCN	#CFCNN	Convert character to numeric	N	FFFFFFFFB1 7CDCB0
2	CVTEFN	#CCCVEN	Convert external form to numeric	N	FFFFFFFFFE 604B90
3	CRTPG	VMXCRTPG	Create program	Y	FFFFFFFFC4 83C300
4	ACTPG	#AICRACT	Activate program	N	FFFFFFFFC2 84B8D0
5	EDIT	.	Edit	N	.
6	REQIO	#SI RQIO1	Request I/O	Y	FFFFFFFFFE F12900
7	MODLUD	#SI LUMD1	Modify logical unit description	Y	FFFFFFFFC4 0D4240
8	MODND	#SI NDMD1	Modify network description	Y	FFFFFFFFC4 A4CB18
9	MODCD	#SI CDMD1	Modify controller description	Y	FFFFFFFFC5 9F8A30
10	MATLUD	#SI LUMT1	Materialize logical unit description	N	FFFFFFFFC5 8AED00
11	MATND	#SI NDMT1	Materialize network description	N	FFFFFFFFC5 8896D0
12	DELDSEN	#DBDELEN	Delete data space entry	Y	FFFFFFFFC4 FEDC50
13	DEACTCR	#DBDACR	Deactivate cursor	Y	FFFFFFFFC5 967D48
14	DESCR	#DBDCR	Destroy cursor	Y	FFFFFFFFC4 FEF458
15	DESDS	#DBDDS	Destroy data space	Y	FFFFFFFFC5 6D0888
16	DESDSINX	#DBDINX	Destroy data space index	Y	FFFFFFFFC4 FF2410
17	ENSDBSEN	#DBENDSE	Ensure data space entries	Y	FFFFFFFFC1 67D070
18	INSDBSEN	#DBINSEN	Insert data space entry	Y	FFFFFFFFB1 66C620
19	MATCRAT	#DBMATCR	Materialize cursor attributes	Y	FFFFFFFFC4 2C4620
20	SETCR	DBSETCUR	Set cursor	Y	FFFFFFFFB3 39EDC0
21	UPDBSEN	#DBUPDEN	Update data space entry	Y	FFFFFFFFFE 7C2340
22	FNDINXEN	.#I I FNDEN	Find independent index entry	N	FFFFFFFFC4 2B3A94
23	INSINXEN	.#I I INSEN	Insert independent index entry	N	FFFFFFFFC5 68FDF8
24	CRTINX	.#I I CRTIX	Create independent index	N	FFFFFFFFC4 1DF9D0
25	RMVINXEN	.#I I RMVEN	Remove independent index entry	N	FFFFFFFFC4 2B3AA8
26	GRANT	#AUGRAU	Grant authority	Y	FFFFFFFFC5 79C058
27	RETRACT	#AURCTAU	Retract authority	Y	FFFFFFFFC5 67DCE8
28	MATAUOBJ	#AUMATOB	Materialize authorized objects	N	FFFFFFFFC5 7094A0
29	MATAUU	#AUMATUU	Materialize authorized users	N	FFFFFFFFC5 9CC040
30	CRTS	#SOCRT	Create space	N	FFFFFFFFC4 2E3130
31	MODS1	#SOMOD	Modify space attributes	N	FFFFFFFFC5 9680B0
32	ACTCR	#DBACR	Activate cursor	Y	FFFFFFFFFE 7CAE30
33	CRTCR	#DBCCR	Create cursor	Y	FFFFFFFFC4 9F50C0
34	RLSDSEN	#DBRLSEN	Release data space entries	Y	FFFFFFFFC2 57CCA0
35	CRCTCX	#MNCRTC	Create context	Y	FFFFFFFFC4 136D78
36	DESCTX	#MNDSTC	Destroy context	Y	FFFFFFFFC5 7F2450
37	RSLVSP	#MNRESSP	Resolve system pointer	N	FFFFFFFFFE 605E98
38	TESTAU	#AUTSTAU	Test authority	N	FFFFFFFFC5 8D6910
39	SIGEV	EMSIGEV	Signal event	Y	FFFFFFFFC2 1CF6C0
R10	MI	Module	Description	B	Address V4R4M0

40	WAITVT	EMWONEVT	Wait on event	Y	FFFFFFFFC2 1D33B0
41	SIGEXCP	#EXSGEXP	Signal exception	N	FFFFFFFFC4 2CD0D0
42	RETEXCPD	#EXRTEXD	Retrieve exception data	N	FFFFFFFFC5 6D12B0
43	RTNEXCP	EXRETURN	Return from exception	N	FFFFFFFFFE 6BA550
44	CRTQ	QMCRTMI Q	Create queue	Y	FFFFFFFFC1 0B8160
45	ENQ	QMENQMI Q	Enqueue	N	FFFFFFFFFE 64A3B0
46	DEQ	QMDEQMI Q	Dequeue	N	FFFFFFFFFE 6DD800
47	MODPRATR	PMO00100	Modify process attributes	Y	FFFFFFFFC2 5C1390
48	INITPR	PMO00097	Initiate process	Y	FFFFFFFFC2 5B7FF0
49	RESPR	PMO00101	Resume process	Y	FFFFFFFFC2 5C86B0
50	SUSPR	PMO00103	Suspend process	Y	FFFFFFFFC2 5CA530
51	MATMOD	VMMMATMO	Materialize module	Y	FFFFFFFFC4 833060
52	CRTAG	RMAGCRAG	Create access group	Y	FFFFFFFFC2 5766B0
53	UNLOCK	RMSL0041	Unlock object	N	FFFFFFFFC2 3D1670
54	XFRLCK	RMSL0026	Transfer object lock	N	FFFFFFFFC2 3D0980
55	DESPG	#PGDESPG	Destroy program	Y	FFFFFFFFC4 321BD4
56	DEACTPG	#AIDACTV	Deactivate program	N	FFFFFFFFC1 672B00
57	RSLVDP	#MNRSLVD	Resolve data pointer	N	FFFFFFFFC5 798350
58	CRTUP	#AUCRTUP	Create user profile	Y	FFFFFFFFC5 9A7830
59	DBMAINT	#DBMAINT	Database maintenance	Y	FFFFFFFFC2 3AC950
60	DESS	#SODES	Destroy space	N	FFFFFFFFC5 631AF0
61	MATS	#SOMAT	Materialize space attributes	N	FFFFFFFFC5 96F2A8
62	SUSOBJ	#SMSUSOB	Suspend object	Y	FFFFFFFFC4 D36750
63	CRTLUD	#SILUCR1	Create logical unit description	Y	FFFFFFFFC4 B16070
64	TESTEXCP	#EXTSTEX	Test exception	N	FFFFFFFFFE 6164D0
65	TESTEVT	EMTSTEVT	Test event	Y	FFFFFFFFC2 1D1A00
66	CANEVTMN	EMCANEVT	Cancel event monitor	Y	FFFFFFFFC2 1C7F50
67	SETDPAT	.	Set data pointer attributes	N	.
68	DESUP	#AUDESUP	Destroy user profile	Y	FFFFFFFFC4 E2E518
69	MATAU	#AUMATAU	Materialize authority	N	FFFFFFFFC4 215020
70	MATUP	#AUMATUP	Materialize user profile	N	FFFFFFFFC6 3BAFB8
71	MODUP	#AUMODUP	Modify user profile	Y	FFFFFFFFC5 5EA7E0
72	XFRO	#AUXFRO	Transfer ownership	Y	FFFFFFFFC4 243408
73	DELPGOBS	#PGDELPO	Delete program observability	Y	FFFFFFFFC5 A3BBD8
74	MATPG	#PGMATPG	Materialize program	N	FFFFFFFFC4 05BFC0
75	MODASA	.	Modify automatic storage allocation	N	.
76	CRTCD	#SICDCR1	Create controller description	Y	FFFFFFFFC5 8B4D10
77	CRTND	#SINDCR1	Create network description	Y	FFFFFFFFC4 A45100
78	DESCD	#SICDDY1	Destroy controller description	Y	FFFFFFFFC4 20AD88
79	DESLUD	#SILUDY1	Destroy logical unit description	Y	FFFFFFFFC5 5EBDC8
80	DESND	#SINDDY1	Destroy network description	Y	FFFFFFFFC5 5DA878
81	MATCD	#SICDMT1	Materialize controller description	N	FFFFFFFFC5 632FA0
82	.	.	.	N	.
83	MATMATR	#RMMATMA	Materialize machine attributes	N	FFFFFFFFC4 F5AA98
84	MODMATR	#RMMODMA	Modify machine attributes	Y	FFFFFFFFC1 5C36B0
85	TERMPR	PMO00105	Terminate process	Y	FFFFFFFFC2 5CC520
86	CRTDS	#DBCDS	Create data space	Y	FFFFFFFFC5 9BBD10
87	CRTDSI NX	#DBCINX	Create data space index	Y	FFFFFFFFC5 4B9B20
88	MATDSAT	#DBMATDS	Materialize data space attributes	Y	FFFFFFFFC5 8F3960
89	MATDSI AT	#DBMATIX	Materialize data space index attributes	Y	FFFFFFFFC5 279AA8
90	RETDSN	#DBRETN	Retrieve data space entry	Y	FFFFFFFFFE 677F40
91	DESI NX	#IIDESIX	Destroy independent index	N	FFFFFFFFC4 26E8C0
92	MATINXAT	#IIMATAT	Materialize independent index attributes	N	FFFFFFFFC5 632BD8
93	CANINVT	.#DOCTRV	Cancel trace invocations	Y	FFFFFFFFC5 5E22D8
94	CANTRINS	#DICTRIN	Cancel trace instructions	Y	FFFFFFFFC5 6EF6D0
95	MATPTRL	#DOMTPTL	Materialize pointer locations	N	FFFFFFFFC5 96EF80
96	MATPTR	#DOMTPTR	Materialize pointer	N	FFFFFFFFFE 70C320
97	MATSOBJ	#DOMTSOB	Materialize system object	N	FFFFFFFFC5 978938
98	TRINV	.#DITRINV	Trace invocations	N	FFFFFFFFC5 5E22C4
99	TRINS	#DITRINS	Trace instructions	Y	FFFFFFFFC4 196878
100	MATCTX	#MNMATC	Materialize context	N	FFFFFFFFC4 053FF8
101	MODADR	#MNMODA	Modify addressability	Y	FFFFFFFFC5 7CFE10
102	RENAME	#MNRENAM	Rename object	Y	FFFFFFFFC5 7BA8E8
R10	MI	Module	Description	B	Address V4R4M0

103	CRTPCS	PM000092	Create process control space	Y	FFFFFFFFC2 5B6350
104	DESPCS	PM000094	Destroy process control space	Y	FFFFFFFFC2 5B7450
105	MATPRATR	PM000098	Materialize process attributes	N	FFFFFFFFFE 6C3870
106	TERMMPR	ITMI TREQ	Terminate machine processing	Y	FFFFFFFFF80 729750
107	CRTDOBJ	#RMC DUPO	Create duplicate object	Y	FFFFFFFFFE 602050
108	DESAG	RMAGDSAG	Destroy access group	Y	FFFFFFFFC2 577150
109	ENSOBJ	#RMENOBJ	Ensure object	N	FFFFFFFFC4 068870
110	MATAGAT	RMAGMTAG	Materialize access group attributes	N	FFFFFFFFC2 5779B0
111	MATRMD	#RMMATD	Materialize resource management data	N	FFFFFFFFC4 284A90
112	MODRMC	#RMMODC	Modify resource management controls	Y	FFFFFFFFC5 5CB888
113	SETACST	#RMSACS	Set access state	N	FFFFFFFFC5 8670F8
114	DBLEVTMN	EMDI SEVT	Disable event monitor	Y	FFFFFFFFC2 1C8960
115	EBLEVTMN	EMENABLE	Enable event monitor	Y	FFFFFFFFC2 1C8FE0
116	MNEVT	EMMONEVT	Monitor event	Y	FFFFFFFFC2 1CA920
117	RETEVTD	EMRETDAT	Retrieve event data	Y	FFFFFFFFC2 1CEF00
118	MODEVETM	EMMODMST	Modify process event mask	N	FFFFFFFFC2 1CA0B0
119	MATEXCPD	#EXMTEXD	Materialize exception description	N	FFFFFFFFC4 0DB970
120	MODEXCPD	#EXMDEXD	Modify exception description	N	FFFFFFFFC1 6AE320
121	DESQ	QMDSTMI Q	Destroy queue	Y	FFFFFFFFC1 0BB230
122	MATQAT	QMMTZATR	Materialize queue attributes	N	FFFFFFFFC1 0BD260
123	MATOBJLK	RMSL0021	Materialize object locks	N	FFFFFFFFC2 3D7410
124	MATPRLLK	RMSL0022	Materialize process locks	N	FFFFFFFFC2 3D9450
125	LOCK	RMSL0054	Lock object	N	FFFFFFFFFE 6CA260
126	MATINV	#DOMATIA	Materialize invocation attributes	N	FFFFFFFFC4 309DB8
127	MATPRMSG	#QSMTPRM	Materialize process message	N	FFFFFFFFC4 113C90
128	MODPRMSG	#QSM DPRM	Modify process message	N	FFFFFFFFC4 3B0420
129	RECLAIM	RCRCMI MI	Reclaim lost objects	Y	FFFFFFFFC5 3F57B0
130	CPYDSE	#DBUCOPY	Copy data space entries	Y	FFFFFFFFC1 A39818
131	INSSDSE	#DBI NSEQ	Insert sequential data space entries	Y	FFFFFFFFFE 7A5C70
132	RETS DSE	#DBRESEQ	Retrieve sequential data space entries	Y	FFFFFFFFFE 7126C0
133	MODPG	#PGMODPG	Modify program	Y	FFFFFFFFC4 280A78
134	LOCKSL	RMSL0018	Lock space location	N	FFFFFFFFFE 69F6B0
135	UNLOCKSL	RMSL0042	Unlock space location	N	FFFFFFFFFE 6A0500
136	MATSELLK	RMSL0024	Materialize selected locks	N	FFFFFFFFC2 3D8750
137	SNSEXCPD	#EXS NEXD	Sense exception description	N	FFFFFFFFC4 318108
138	SNDPRMSG	EXSNDPM	Send process message	N	FFFFFFFFFE 6BBCA0
139	APYJCHG	#JOAPPLY	Apply journaled changes	Y	FFFFFFFFC5 918C80
140	CRTJP	#JOCRTJP	Create journal port	Y	FFFFFFFFC4 0BB9A0
141	CRTJS	#JOCRTJS	Create journal space	Y	FFFFFFFFC5 797AF8
142	DESJP	#JODESJP	Destroy journal port	Y	FFFFFFFFC5 1B11E0
143	DESJS	#JODESJS	Destroy journal space	Y	FFFFFFFFC4 315D30
144	JRNLD	#JOURDAT	Journal data	Y	FFFFFFFFC1 5A2FE0
145	JRNLOBJ	#JOJOB	Journal object	Y	FFFFFFFFC5 695B40
146	MATJPAT	#JOMATJP	Materialize journal port attributes	N	FFFFFFFFC1 6B9DB8
147	MATJSAT	#JOMATJS	Materialize journal space attributes	N	FFFFFFFFC1 56A880
148	MATJOAT	#JOMATOA	Materialize journaled object attributes	N	FFFFFFFFC1 5C8C60
149	MATJOB	#JOMATJO	Materialize journaled objects	N	FFFFFFFFC5 9B3910
150	MODJP	#JOMODJP	Modify journal port	Y	FFFFFFFFC6 5DF858
151	RETJENT	#JORETEN	Retrieve journal entries	Y	FFFFFFFFC5 521388
152	COMMIT	#COCOMIT	Commit	Y	FFFFFFFFFE 7C97E8
153	CRTCB	#COCRCOB	Create commit block	Y	FFFFFFFFC4 01DC60
154	DECOMMIT	#CODCOMIT	Decommit	Y	FFFFFFFFC4 878118
155	DESCB	#CODSCOB	Destroy commit block	Y	FFFFFFFFC4 FFCCB0
156	MATCBATR	#COMACOB	Materialize commit block attributes	N	FFFFFFFFC4 3225A8
157	MODCB	#COMOCOB	Modify commit block	Y	FFFFFFFFC4 FFD438
158	SETIEXIT	.	Set invocation exit	N	.
159	CLRIEXIT	.	Clear invocation exit	N	.
160	CANINV	#EXCANIN	Cancel invocation	N	FFFFFFFFC5 9F2030
161	CRTBPGM	VBNCRTBP	Create bound program	Y	FFFFFFFFC4 8399E0
162	MODINX	#IIMODIX	Modify independent index	N	FFFFFFFFC4 13D288
163	MODSOBJ	#DOMODSO	Modify system object	Y	FFFFFFFFF4 06D6D8
164	MATAOL	RMSL0019	Materialize allocated object locks	N	FFFFFFFFC2 3D6850
165	MATINAT	#DOMTINS	Materialize instruction attributes	N	FFFFFFFFC5 821788
R10	MI	Module	Description	B	Address V4R4M0

166	MODDSIA	#DBMODIX	Modi fy data space index attributes	Y	FFFFFFFFC5 A2E2B0
167	GRNTLI KE	#AUGRLAU	Grant like authority	Y	FFFFFFFFC5 80D1D8
168	MATEVTMN	EMMTZMON	Materialize event monitors	N	FFFFFFFFC2 1C9C50
169	WAITTIME	PMO00106	Wait on time	N	FFFFFFFFC2 5CDED0
170	MATDRECL	RMSL0020	Materialize data space record locks	N	FFFFFFFFC2 3DAFC0
171	MATPRECL	RMSL0023	Materialize process record locks	N	FFFFFFFFC2 3DC2A0
172	MATQMSG	QMMTZMSG	Materialize queue messages	N	FFFFFFFFC1 0BD670
173	RESAG	RMAGRSAG	Reset access group	Y	FFFFFFFFC2 577720
174	MATINVS	#DOMTSTK	Materialize invocation stack	N	FFFFFFFFC1 66DFF0
175	CRTQS	#QSCRTQS	Create queue space	Y	FFFFFFFFC5 747E90
176	DESQS	#QSDSQS	Destroy queue space	Y	FFFFFFFFC4 2D6FF0
177	MATQSAT	#QSMTQSA	Materialize queue space attributes	Y	FFFFFFFFC5 A01138
178	MODQSAT	#QSMQSA	Modi fy queue space attributes	Y	FFFFFFFFC4 277BE0
179	MATBPGM	VBNMATBP	Materialize bound program	N	FFFFFFFFC4 82D370
180	REQPO	#TPVPASD	Request path operation	Y	FFFFFFFFC1 4C3928
181	LOGICAL	.	Logical AND, OR, EXCLUSIVE OR	N	.
182	MODDSAT	#DBMDSAT	Modi fy data space attributes	Y	FFFFFFFFC5 9416C8
183	MODINVAT	#DOMDI NA	Modi fy invocation attributes	N	FFFFFFFFC5 5DC048
184	ESTDSI KR	#DBIXGES	Estimate size of DSI key range	Y	FFFFFFFFC5 383D98
185	CRTDMPS	#LDCRTDS	Create dump space	Y	FFFFFFFFC5 632238
186	DESDMPS	#LDDSDS	Destroy dump space	Y	FFFFFFFFC4 200AD8
187	INSNDMPD	#LDINSDD	Insert dump data	Y	FFFFFFFFC5 A5E4D0
188	MATDMPS	#LDMATDS	Materialize dump space	N	FFFFFFFFC5 96E870
189	MODDMPS	#LDMODDS	Modi fy dump space	Y	FFFFFFFFC4 0DC6B0
190	RETDMPD	#LDRETDD	Retrieve dump data	Y	FFFFFFFFC4 0871E0
191	MODMOD	VMMMODMO	Modi fy module	Y	FFFFFFFFC4 82C710
192	TERMINST	PMO00104	Terminate instruction	Y	FFFFFFFFC2 5CBB60
193	CIPHER	#CRCI PHR	Cipher	N	FFFFFFFFC1 3FCD50
194	MODBPGM	VBNMDBBP	Modi fy bound program	Y	FFFFFFFFC4 82B590
195	CRTCSO	#SICSCR1	Create class of service description	Y	FFFFFFFFC5 6BE398
196	CRTMD	#SIMDCR1	Create mode description	Y	FFFFFFFFC5 870ED0
197	DESCSO	#SICSDY1	Destroy class of service description	Y	FFFFFFFFC5 A5DA78
198	DESMD	#SIMDDY1	Destroy mode description	Y	FFFFFFFFC4 1C1610
199	MATCSO	#SICSMT1	Materialize class of service description	N	FFFFFFFFC4 29CAE8
200	MATMD	#SIMDMT1	Materialize mode description	N	FFFFFFFFC5 5E3088
201	MODCSO	#SICSMO1	Modi fy class of service description	Y	FFFFFFFFC5 7EEA08
202	MODMD	#SIMDMO1	Modi fy mode description	Y	FFFFFFFFC5 9C8898
203	REQSO	#SI RQSO1	Request statistics operation	Y	FFFFFFFFC1 625258
204	MATMATR2	#RMMATMA	Materialize machine attributes 2	N	FFFFFFFFC4 F5AA98
205	CRTAL	#AUCRTAL	Create authorization list	Y	FFFFFFFFC4 0ADB48
206	DESAL	#AUDESAL	Destroy authorization list	Y	FFFFFFFFC5 86DC98
207	MATAL	#AUMATAL	Materialization authorization list	N	FFFFFFFFC4 3A80A0
208	MODAL	#AUMODAL	Modi fy authorization list	Y	FFFFFFFFC5 783A90
209	CPYECLAP	.	Copy extended chars left adj. with pad	N	.
210	XLATWTDS	.	Translate with table and DBCS skip	N	.
211	CRTDCT	#DSCRTD	Create dictionary	Y	FFFFFFFFC5 9F5048
212	DESDCT	#OEDSDC	Destroy dictionary	Y	FFFFFFFFC5 8B1B88
213	CHKDCT	#DSLEXAM	Check dictionary	Y	FFFFFFFFC5 7A8EB8
214	MATDCT	#OEMATDC	Materialize dictionary attributes	Y	FFFFFFFFC5 5B9D50
215	GRNTLI KO	#AUGRLOB	Grant like object authority	Y	FFFFFFFFC4 101738
216	EXTRMOD	VBNEXTMO	Extract module	Y	FFFFFFFFC4 82AA50
217	DIAG	#RI DI AGF	Diagnose	Y	FFFFFFFFC4 1F4378
218	CLSBSS	.	Close byte-string space	Y	.
219	CRTBSS	.	Create byte-string space	Y	.
220	DESBSS	#BSSDES	Destroy byte-string space	Y	FFFFFFFFC5 7A48F0
221	MATBSS	#BSSMAT	Materialize byte-string space	Y	FFFFFFFFC4 06EF38
222	MODBSS	.	Modi fy byte-string space	Y	.
223	OPNBSS	.	Open byte-string space	Y	.
224	RETBSS	.	Retrieve byte-string	Y	.
225	STBSS	.	Store byte-string	Y	.
226	MODBSLS	.	Modi fy byte-string lock state	Y	.
227	DBGINT	.	Debug interpreter	Y	FFFFFFFFC4 7E65A0
228	TESTEAU	#AUTSTPS	Test extended authorities	N	FFFFFFFFC5 8989A0
R10	MI	Module	Description	B	Address V4R4M0

229	CRTNWID	#SIPGCR1	Create network interface description	Y	FFFFFFFFC4 A54350
230	MATNWID	#SIPGMT1	Materialize network interface description	N	FFFFFFFFC4 116570
231	MODNWID	#SIPGMD1	Modify network interface description	Y	FFFFFFFFC5 6A98C8
232	DESNWID	#SIPGDY1	Destroy network interface description	Y	FFFFFFFFC5 8F1A28
233	CRTCNNL	#SICLCR1	Create connection list	Y	FFFFFFFFC4 13C7D8
234	MATCNNL	#SICLMT1	Materialize connection list	N	FFFFFFFFC5 961DA8
235	MODCNNL	#SICLMD1	Modify connection list	Y	FFFFFFFFC4 2B0F40
236	DESCNNL	#SICLDY1	Destroy connection list	Y	FFFFFFFFC5 69D418
237	SRCHDCT	#DSTWAIN	Search directory	Y	FFFFFFFFC4 0F7028
238	INCD	.#CFINCD	Increment date	N	FFFFFFFFFE 6262F0
239	DECD	.#CFDECD	Decrement date	N	FFFFFFFFFE 626304
240	INCT	.#CFINCT	Increment time	N	FFFFFFFFFE 613E50
241	DECT	.#CFDECT	Decrement time	N	FFFFFFFFFE 613E64
242	INCTS	.#CFINCTS	Increment timestamp	N	FFFFFFFFFE 7543E0
243	DECTS	.#CFDECTS	Decrement timestamp	N	FFFFFFFFFE 7543F4
244	CDD	.#CFCDD	Compute date duration	N	FFFFFFFFFE 626318
245	CTD	.#CFCTD	Compute time duration	N	FFFFFFFFFE 613E78
246	CTSD	.#CFCTSD	Compute timestamp duration	N	FFFFFFFFFE 754408
247	CVTD	.#CFCVTD	Convert date	N	FFFFFFFFFE 6262DC
248	CVTT	.#CFCVTT	Convert time	N	FFFFFFFFFE 613E3C
249	CVTTS	.#CFCVTTS	Convert timestamp	N	FFFFFFFFFE 7543CC
250	MODCTX	#MNMDC	Modify context	Y	FFFFFFFFC5 865AB0
251	CPRDATA	#CFCPRDT	Compress data	N	FFFFFFFFC5 6F9D20
252	DCPDATA	#CFDCPDT	Decompress data	N	FFFFFFFFC5 76FF8
253	CRTMOD	VMMCRTMO	Create module	Y	FFFFFFFFC4 83B380
254	DESMOD	VMMDESMO	Destroy module	Y	FFFFFFFFC4 822BA0
255	CRTHS	#HMCRTMI	Create heap space	N	FFFFFFFFC5 629178
256	DESHs	#HMDESMI	Destroy heap space	N	FFFFFFFFC4 1F1F00
257	FREHSSMK	#HMFMKMI	Free heap space storage from mark	N	FFFFFFFFC5 8EC418
258	MATHSAT	#HMMATMI	Materialize heap space attributes	N	FFFFFFFFC5 9666E8
259	SETHSSMK	#HMMRKMI	Set heap space storage mark	N	FFFFFFFFC5 6227B0
260	DESAGP	#AIDSGAP	Destroy activation group	N	FFFFFFFFC2 84C500
261	MATACTAT	#AIDACTA	Materialize activation attributes	N	FFFFFFFFC1 4ED1D0
262	MATAGPAT	#AIDAGPA	Materialize activation group attributes	N	FFFFFFFFC1 68D8F0
263	MATPRAGP	#AIDMPAGP	Materialize process activation groups	N	FFFFFFFFC1 660DC0
264	.	#PODLNK	#PODLNK	Y	FFFFFFFFB1 7C8B30
265	.	#PODULNK	#PODULNK	Y	FFFFFFFFB1 7C61C0
266	.	#PODMATL	#PODMATL	Y	FFFFFFFFB1 7CB400
267	.	#PODLOOK	#PODLOOK	Y	FFFFFFFFB1 7C5600
268	.	#DBRETSM	#DBRETSM	Y	FFFFFFFFC2 409D28
269	.	#DBCRTIM	#DBCRTIM	Y	FFFFFFFFC4 9E7B18
270	XFERGO	#AUXFRGO	Transfer group owner	Y	FFFFFFFFC5 5F3020
271	TESTULA	#AUTSTUL	Test user list authority	N	FFFFFFFFC5 831300
272	MATUPI D	#AUMATID	Materialize user profiles from ID	N	FFFFFFFFC1 3D7698
273	CRTMTX	MASOCRMM	Create mb mutex	N	FFFFFFFFB1 6161A0
274	DESMTX	MASODSMM	Destroy mb mutex	N	FFFFFFFFC1 3485A0
275	.	.	.	N	.
276	.	.	.	N	.
277	.	#DOTRCPR	#DOTRCPR	Y	FFFFFFFFC5 674340
278	.	#SISVCR1	#SISVCR1	Y	FFFFFFFFC4 0CA500
279	.	#SISVMT1	#SISVMT1	Y	FFFFFFFFC5 8AA8D0
280	.	#SISVMD1	#SISVMD1	Y	FFFFFFFFC4 2470E8
281	.	#SISVDY1	#SISVDY1	Y	FFFFFFFFC5 5E9DC8
282	ACTBPGM	#AIDACTBP	Activate bound program	N	FFFFFFFFC2 849A00
283	DEACTBM	#AIDACTM	Deactivate bound module	Y	FFFFFFFFC1 402D70
284	RINZSTAT	#AIDRSTAT	Re-initialize static storage	N	FFFFFFFFC1 58ADA0
285	TESTSOBJ	#DOTSSOB	Test system object	Y	FFFFFFFFC4 0584D8
286	MATPTRIF	#DOMTPTI	Materialize pointer information	N	FFFFFFFFC4 295AF0
287	.	#AUTSTSP	#AUTSTSP	Y	FFFFFFFFC5 8B45A0
288	.	#PODMODD	#PODMODD	Y	FFFFFFFFC1 3BEA28
289	.	#PODRNM	#PODRNM	Y	FFFFFFFFC1 5C9F70
290	.	#AIDMSSAT	#AIDMSSAT	Y	FFFFFFFFC1 523CC0
291	.	#DBPTMCH	#DBPTMCH	Y	FFFFFFFFC4 1294E8
R10	MI	Module	Description	B	Address V4R4M0

292	.	.	.	N	.
293	.	.	.	N	.
294	.	.	.	N	.
295	.	.	.	N	.
296	.	.	.	N	.
297	.	.	.	N	.
298	.	.	.	N	.
299	.	.	.	N	.
300	MATIN VAT	#DOMTINA	Materialize invocation attributes	N	FFFFFFFFC1 5D89C0
301	MATIN VIF	#DOMTIVF	Materialize invocation information	N	FFFFFFFFFE 70B9B8
302	FNDRI NVN	#DOFN DIN	Find relative invocation number	N	FFFFFFFFC1 3E2E80
303	SRCHHRI	IOMI 0117	Search HRI	Y	FFFFFFFFC1 23CB70
304	DESHRI	IOMI 0078	Destroy HRI	Y	FFFFFFFFC1 B05A20
305	DESI TER	IOMI 0080	Destroy iterator	Y	FFFFFFFFC2 6ADD70
306	MATHRI R	IOMI 0101	Materialize HRI record	Y	FFFFFFFFC2 51D230
307	MATHRI	IOMI 0097	Materialize HRI	Y	FFFFFFFFC2 8B8B00
308	CRTI TER	IOMI 0074	Create iterator	Y	FFFFFFFFC2 25D130
309	MODI TER	IOMI 0104	Modify HRI	Y	FFFFFFFFC2 8BCAE0
310	MATHRI CD	IOMI 0099	Materialize HRI code	Y	FFFFFFFFC2 8B9600
311	.	.	.	N	FFFFFFFFFE 6DD820
312	.	.	.	N	FFFFFFFFC5 9680C4
313	.	.	.	N	.
314	?	#AISFIAG	#AISFIAG	N	FFFFFFFFFE 606AE0
315	SIGEXCP	#MXSIGEX	Signal exception	N	FFFFFFFFC5 A77258
316	?	#AITSBAL	#AITSBAL	N	FFFFFFF B1 6367C0
317	MATSSP	#SOMATSS	Materialize Secondary Spaces	N	FFFFFFFFC1 49EB08
318	CVTCS	RJE00014	Convert character to SNA	N	FFFFFFFFC2 080B80
319	CVTSC	RJE00015	Convert SNA to character	N	FFFFFFFFC2 07F5F0
320	CVTCH	RJE00007	Convert character to hex	N	FFFFFFFFC2 07DEA0
321	CVTMC	RJE00008	Convert MRJE to character	N	FFFFFFFFC2 07EB10
322	NFYHDWCH	IOMI 0107	IOMI NFYHDWCH	Y	FFFFFFFFC2 9370C0
323	.	.	.	Y	FFFFFFFFC2 9013A0
324	MATCFGD	IOMI 0094	Materialize configuration description	Y	FFFFFFFFC2 9000C0
325	VALCFGD	IOMI 0125	Validate configuration description	Y	FFFFFFFFC2 90E230
326	CRTCFGD	IOMI 0077	Create configuration description	Y	FFFFFFFFC2 8FB020
327	MODCFGD	IOMI 0103	Modify configuration description	Y	FFFFFFFFC2 9038A0
328	.	.	.	N	.
329	.	.	.	N	.
330	.	.	.	N	.
331	.	.	.	N	FFFFFFFFC2 910950
332	MATPUI D	PMO00109	Materialize process UID GUID	Y	FFFFFFFFC2 5BF600
333	?	#PODNOOP	#PODNOOP	N	FFFFFFFFC1 5DF6E0
334	SETOBPFP	XOMSETOP	Set object pointer from pointer	N	FFFFFFFFC2 9664A0
335	.	.	.	Y	.
336	.	.	.	Y	.
337	DESSOM	XOMDESOM	Destroy SOM object	Y	FFFFFFFFC2 9662E0
338	.	#XOMSOP	#XOMSOP	Y	FFFFFFFFC5 33DEE0
339	?	CCCLRSIZ	Clear def size exception	N	FFFFFFFFC2 4D4020
340	?	MASOMAT	Materialize MB Mutex	N	FFFFFFFFC1 337EC0
341	MATPRMTX	MASOMATP	Materialize process MB Mutex	N	FFFFFFFFC1 339160
342	.	#CRCPHY	#CRCPHY	Y	FFFFFFFFC1 6BBCE0
343	.	#POFSRT	#POFSRT	Y	FFFFFFFFC4 FDCFB8
344	FSERVOP	#FSERVOP	#FSERVOP	Y	FFFFFFFFC1 ADC858
345	.	#POFVRD	#POFVRD	Y	FFFFFFFFC4 AC4438
346	.	.	.	Y	FFFFFFFFC4 AC5DF4
347	?	#AIMTAX	#AIMTAX	N	FFFFFFFFC2 84E700
348	.	#POFSELM	#POFSELM	Y	FFFFFFFFC5 64CDD8
349	GENUUI D	AUGNUUI D	Generate UUID	N	FFFFFFFFC2 631840
350	.	.	.	N	.
351	MATPNSI G	PXSGMTPN	PXSGMATPNSI G	N	FFFFFFFFC5 45E690
352	MATSI GMN	PXSGMTMN	PXSGMATSI GMN	N	FFFFFFFFC5 45E820
353	MODPSI GM	PXSGMDPM	PXSGMODPSI GM	N	FFFFFFFFC5 45EEC0
354	MODSI GBM	PXSGMDBM	PXSGMODSI GBM	N	FFFFFFFFC5 45F100
R10	MI	Module	Description	B	Address V4R4M0

355	MODSI GMN	PXSGMDMN	PXSGMODSI GMN	N	FFFFFFFFC5 45F2A0
356	SIGT I MER	PXSGT I ME	Signal timer	N	FFFFFFFFC5 45F960
357	SNDSI G	PXSGSND	Send signal	N	FFFFFFFFC5 45FAF0
358	WAITSI G	PXSGWAIT	Wait for signal	N	FFFFFFFFC5 45FC20
359	MATPI D	PXPMMATP	Materialize process identifiers	N	FFFFFFFFC2 277250
360	MODPI D	PXPMMODP	Modify process identifiers	N	FFFFFFFFC2 27B210
361	?	CCSETSI Z	Set def size exception	N	FFFFFFFFC2 4D4080
362	MATMPRI	PMO00116	Materialize Machine Processor Info	Y	FFFFFFFFC2 5BEE80
363	UNLOCKSL	RMSLULOL	Unlock object locations	N	FFFFFFFFC2 3CED50
364	LOCKOL	RMSLLKOL	Lock object locations	N	FFFFFFFFC2 3CF370
365	MODUI DBS	AUMI DBS	Modify UID base	Y	FFFFFFFFC2 631C10
366	EXI TTH	PMEXI TTH	Exit thread	N	FFFFFFFFC2 5B7980
367	I NI TTH	PMI NI TTH	Initiate thread	N	FFFFFFFFC2 5BB6C0
368	MATTHI F	PMMATI F	Materialize thread information	N	FFFFFFFFC2 5C0BE0
369	MODTH	PMMDTA	Modify thread attributes	N	FFFFFFFFC2 5C5A00
370	OPNTH	PMOPENTH	Open thread	N	FFFFFFFFC2 5C77D0
371	RESTH	PMRESTH	Resume thread	N	FFFFFFFFC2 5C8FB0
372	.	.	.	N	.
373	.	.	.	N	.
374	RETHSTS	PMRTHSTS	Retrieve thread status	N	FFFFFFFFC2 5C9A80
375	SUSTH	PMSUSTH	Suspend thread	N	FFFFFFFFC2 5CB090
376	TERMTH	PMTERMTH	Terminate thread	N	FFFFFFFFC2 5CD2F0
377	.	.	.	Y	FFFFFFFFFE 6ED890
378	.	.	.	N	FFFFFFFFC2 75B430
379	CRTHMTX	MASOCRTH	Create handle Mutex	N	FFFFFFFFC1 332360
380	OPNHMTX	MASOOPNM	Open handle Mutex	N	FFFFFFFFC1 33BBA0
381	CRTCND	MASOCRTC	Create condition	N	FFFFFFFFC1 348270
382	OPNCND	MASOOPNC	Open condition	N	FFFFFFFFC1 34B8E0
383	SETCND	MASOSETC	Set condition	N	FFFFFFFFC1 34EBB0
384	RESETCND	MASORESC	Reset condition	N	FFFFFFFFC1 34E9F0
385	PULSECND	MASOPULC	Pulse condition	N	FFFFFFFFC1 34E830
386	CLSHND	MACLSHND	Close handle	N	FFFFFFFFC2 7E6930
387	TESTKEY	MATSTKEY	Test key	N	FFFFFFFFC2 7E6D10
388	RETHNDT	MARTHNDT	Retrieve handle type	N	FFFFFFFFC2 7E6980
389	?	#HMCRTHB	CRTHB	N	FFFFFFFFC4 0E63B0
390	?	#HMMATHB	MATHB	N	FFFFFFFFC5 5C24A0
391	.	AUAPYRST	Apply user RST	Y	FFFFFFFFC2 6300A0
392	?	JVASCVOC	Java deto object create scv	N	FFFFFFFFFE 635080
393	.	.	.	N	FFFFFFFFFE 635420
394	.	.	.	N	FFFFFFFFFE 670610
395	.	.	.	N	FFFFFFFFFE 670640
396	.	.	.	N	FFFFFFFFFE 670670
397	.	.	.	N	FFFFFFFFFE 6709C0
398	.	.	.	N	FFFFFFFFFE 670AB0
399	.	.	.	N	FFFFFFFFFE 670B00
400	.	.	.	N	FFFFFFFFFE 70DF00
401	*	CFMVLI CO	CFMVLI COPT	Y	FFFFFFFFC1 BAFBE0
402	.	.	.	N	FFFFFFFFC6 1A33A0
403	.	JATRFORM	Java Transform	Y	FFFFFFFFC5 64CAFO
404	.	.	.	Y	FFFFFFFFC6 0B85D0
405	?	JVACRTVM	Java Create JVM	N	FFFFFFFFC5 5CA3F0
406	?	JVADESVM	Java Destroy JVM	N	FFFFFFFFC5 167AB0
407	.	.	.	N	FFFFFFFFC6 1A2FE0
408	.	.	.	Y	FFFFFFFFC2 052D30
409	.	.	.	N	FFFFFFFFC2 0504F0
410	.	PDCREMI	PDCRECORDMI EVENT	Y	FFFFFFFFC1 38C9E0
411	.	CSTE0008	CSTECLUEOP	N	FFFFFFFFC2 752920
412	.	MASOCRPS	Create NB semaphore	N	FFFFFFFFC1 3F2CF0
413	.	MASODPSM	MASODESTROYPBSEMAPHORE	N	FFFFFFFFC1 3278A0
414	.	MASO0SEM	Open NB semaphore	N	FFFFFFFFC2 70B010
415	.	MASOCNSM	Close NB semaphore	N	FFFFFFFFC1 347B80
416	.	MASOULSM	Unlink NB semaphore	N	FFFFFFFFC1 34ED70
417	.	MASOMAPS	MASOMATPROCSEMAPHORE	N	FFFFFFFFC1 349A00
R10	MI	Module	Description	B	Address V4R4M0



418	.	MASOMATS	Materialize semaphore	N	FFFFFFFFC1 34A830
419	SOCKETOP	LOSOCKOP	LOSOCKETOP	Y	FFFFFFFFC4 FAC5B0
420	FSOP	POLFSOP	POLFSOP	Y	FFFFFFFFB1 70CF90
421	DMOP	POLDMOP	POLDMOP	Y	FFFFFFFFB1 65F4E0
422	CRTRGN	PMCRTGRN	Create region	Y	FFFFFFFFC2 5D8330
423	DESRGN	PMDESRGN	Destroy region	N	FFFFFFFFC2 5D8D50
424	MATRGNAT	PMMATREG	Materialize region attributes	N	FFFFFFFFC2 5D99A0
425	CRTMOBJ	PMCRTMOB	Create memory object	N	FFFFFFFFC1 AF23E0
426	DESMOJB	PMDESMOJB	Destroy memory object	N	FFFFFFFFC1 AE61F0
427	MATMOBJ	PMMATMOB	Materialize memory object	N	FFFFFFFFC2 5D94B0
428	ALCMEM	PMALCMEM	Allocate memory	N	FFFFFFFFC2 5D74E0
429	DEALLOCA	PMDALMEM	Deallocate memory	N	FFFFFFFFC2 5D8550
430	MAPMEM	PMMAPMEM	Map memory	N	FFFFFFFFC2 5D9020
431	REMAPMM	PMRMPMEM	Remap memory	N	FFFFFFFFC2 5D9DD0
432	.	PMMDYMP	Defy memory protection !?	N	FFFFFFFFC2 5D9A10
433	MATMEMAT	PMMATMEM	Materialize memory attributes	N	FFFFFFFFC2 5D9960
434	.	PMRTVOTY	Retrieve address of space object type	N	FFFFFFFFC2 5DA220
435	.	.	.	Y	FFFFFFFFC4 85AFA0
436	.	.	.	Y	FFFFFFFFC4 857610
437	.	.	.	Y	FFFFFFFFC4 855AF0
438	.	.	.	Y	FFFFFFFFC4 089450
439	.	MASOCLTK	Clear token	N	FFFFFFFFC1 346B30
440	.	.	.	N	FFFFFFFFFE 6C9A90
441	.	#DBCPYFL	Copy file	Y	FFFFFFFFC1 2224A8
442	.	#DBCMPFL	Compare files	Y	FFFFFFFFC2 4979C0

## Appendix F

### Open and I/O Feedback Areas

#### Open Feedback Area

The system keeps track of the status of a file in feedback areas once it is successfully opened. As operations are performed on a file, these feedback areas are updated to reflect the latest status. These feedback areas give you greater control over applications and provide important information when errors occur. The feedback areas are established at open time, and there is one feedback area for each open file. An exception is for shared files, where shared feedback areas are used, as well as the shared data path between the programs sharing the file and the file itself.

The open feedback area is the part of the open data path (ODP) that contains general information about the file after it has been opened. It also contains file-specific information, depending on the file type, along with information about each device or communications session defined for the file. This information is set during open processing and may be updated as other operations are performed.

Position	Data Type	Contents	File Types
1	CHAR(2)	Open data path (ODP) type:	All
		DS    Display, tape, ICF, save, printer file not being spooled, or diskette file not being spooled	
		DB    Database member	
		SP    Printer or diskette file being spooled or inline data file	
3	CHAR(10)	Name of the file being opened. If the ODP type is DS, this is the name of the device file or save file. If the ODP type is SP, this is the name of the device file or the inline data file. If the ODP type is DB, this is the name of the database file that the member belongs to	All
13	CHAR(10)	Name of the library containing the file. For an inline data file, the value is *N.	All
23	CHAR(10)	Name of the spooled file. The name of a database file containing the spooled input or output records.	Printer or diskette being spooled or inline data
33	CHAR(10)	Name of the library in which the spooled file is located.	Printer or diskette being spooled or inline data
43	BIN(2)	Spooled file number	Printer or diskette being spooled
45	BIN(2)	Maximum record length	All
47	BIN(2)	Maximum key length	Database
49	CHAR(10)	Member name: ° If ODP type is DB, the member name in the file named at position 3. If file is overridden to MBR(*ALL), the member name that supplied the last record. ° If ODP type is SP, the member name in the file named at position 23	Database, printer, diskette, inline data
59	BIN(4)	Reserved	

Position	Data Type	Contents	File Types
63	BIN(4)	Reserved	
67	BIN(2)	File type: 1 Display 2 Printer 4 Diskette 9 Save 10 DDM 11 ICF 20 Inline data 21 Database	All
69	CHAR(3)	Reserved	
72	BIN(2)	Number of lines on a display screen or number of lines on a printed page Length of the null field byte map.	Display, printer Database
74	BIN(2)	Number of positions on a display screen or number of characters on a printed line. Length of the null key field byte map.	Display, printer Database
76	BIN(4)	Number of records in the member at open time. For a join logical file, the number of records in the primary. Supplied only if the file is being opened for input.	Database, inline data
80	CHAR(2)	Access type: AR Arrival sequence. KC Keyed with duplicate keys allowed. Duplicate keys are accessed in first-changed-first-out (FCFO) order KF Keyed with duplicate keys allowed. Duplicate keys are accessed in first-in-first-out (FIFO) order KL Keyed with duplicate keys allowed. Duplicate keys are accessed in last-in-first-out (LIFO) order KN Keyed with duplicate keys allowed. The order in which duplicate keys are accessed can be one of the following: ◦ First-in-first-out (FIFO) ◦ Last-in-first-out (LIFO) ◦ First-changed-first-out (FCFO) KU Keyed, unique.	Database
82	CHAR(1)	Duplicate key indication. Set only if the access path is KC, KF, KL, KN, or KU: D Duplicate keys allowed if the access path is KF or KL U Duplicate keys are not allowed; all keys are unique and the access path is KU	Database
83	CHAR(1)	Source file indication. Y File is a source file N File is not a source file	Database, tape, diskette, inline data
84	CHAR(10)	Reserved	
94	CHAR(10)	Reserved	
104	BIN(2)	Offset to volume label fields of open feedback area.	Diskette, tape
106	BIN(2)	Maximum number of records that can be read or written in a block when using blocked record I/O	All
108	BIN(2)	Overflow line number.	Printer

Bit numbering is from 0 and up, with bit 0 being the most significant bit.

Position	Data Type	Contents	File Types
110	BIN(2)	Blocked record I/O record increment. Number of bytes that must be added to the start of each record in a block to address the next record in the block	All
112	BIN(4)	Reserved	
116	CHAR(1)	Miscellaneous flags	
		Bit 0: Reserved	
		Bit 1: File shareable	All
		0 File was not opened shareable	
		1 File was opened shareable (SHARE(*YES))	
		Bit 2: Commitment control	All
		0 File is not under commitment control	
		1 File is under commitment control.	
		Bit 3: Commitment lock level	Database
		0 Only changed records are locked (LCKLVL (*CHG)). If this bit is zero and bit 7 of the character at position 133 is one, then all records accessed are locked, but the locks are released when the current position in the file changes (LCKLVL (*CS)).	
		1 All records accessed are locked (LCKLVL (*ALL)).	
		Bit 4: Member type	Database
		0 Member is a physical file member	
		1 Member is a logical file member	
		Bit 5: Field-level descriptions	All, except database
		0 File does not contain field-level descriptions	
		1 File contains field-level descriptions	
		Bit 6: DBCS or graphic-capable file	Database, display, printer, tape, diskette, ICF
		0 File does not contain DBCS or graphic-capable fields	
		1 File does contain DBCS or graphic-capable fields	
		Bit 7: End-of-file delay	Database
		0 End-of-file delay processing is not being done	
		1 End-of-file delay processing is being done	
117	CHAR(10)	Name of the requester device. For display files, this is the name of the display device description that is the requester device. For ICF files, this is the program device name associated with the remote location of *REQUESTER. This field is supplied only when either a device or remote location name of *REQUESTER is being attached to the file by an open or acquire operation. Otherwise, this field contains *N	Display, ICF
127	BIN(2)	File open count. If the file has not been opened shareable, this field contains the value 1. If the file has been opened shareable, this field contains the number of programs currently attached to this file	All
129	BIN(2)	Reserved	
131	BIN(2)	Number of based-on physical members opened. For logical members, this is the number of physical members over which the logical member was opened. For physical members, this field is always set to 1	Database

Position	Data Type	Contents	File Types
133	CHAR(1)	Miscellaneous flags:	
		Bit 0: Multiple member processing	Database
		0 Only the member specified will be processed	
		1 All members will be processed	
		Bit 1: Join logical file	Database
		0 File is not a join logical file	
		1 File is a join logical file	
		Bit 2: Local or remote data	Database
		0 Data is stored on local system	
		1 Data is stored on remote system	
		Bit 3: Remote S/38, AS/400, or iSeries data. Applicable only if the value of Bit 3 is 1	Database
		0 Data is on a remote S/38, AS/400, or iSeries system.	
		1 Data is not on a remote S/38, AS/400, or iSeries system	
		Bit 4: Separate indicator area	Printer, display, ICF
		0 Indicators are in the I/O buffer of the program	
		1 Indicators are not in the I/O buffer of the program. The DDS keyword, INDARA, was used when the file was created	
		Bit 5: User buffers	All
		0 System creates I/O buffers for the program	
		1 User program supplies I/O buffers	
134	CHAR(2)	Bit 6: Reserved.	
		Bit 7: Additional commitment lock level indicator. This is only valid if bit 2 of the character at position 115 is one.  If bit 3 of the character at position 116 is zero:	Database
		0 Only changed records are locked (LCKLVL(*CHG))	
		1 All records accessed are locked, but the locks are released when the current position in the file changes (LCKLVL(*CS))	
		If bit 3 of the character at position 116 is one:	
		0 All records accessed are locked (LCKLVL(*ALL))	
		1 Reserved	
		Open identifier. This value is unique for a full open operation (SHARE(*NO)) or the first open of a file that is opened with SHARE(*YES). This is used for display and ICF files, but is set up for all file types. It allows you to match this file to an entry on the associated data queue	All
136	BIN(2)	The field value is the maximum record format length, including both data and file-specific information such as: first-character forms control, option indicators, response indicators, source sequence numbers, and program-to-system data. If the value is zero, then use the field at position 45	Printer, diskette, tape, ICF
138	BIN(2)	Coded character set identifier (CCSID) of the character data in the buffer	Database

Position	Data Type	Contents	File Types
140	CHAR(1)	Miscellaneous flags	
		Bit 0: Null-capable field file	Database
		0 File does not contain null-capable fields	
		1 File contains null-capable fields	
		Bit 1: Variable length fields file	Database
		0 File does not contain any variable length fields	
		1 File contains variable length fields	
		Bit 2: Variable length record processing	Database
		0 Variable length record processing will not be done	
		1 Variable length record processing will be done	
		Bit 3: CCSID character substitution	Database, Display
		0 No substitution characters will be used during CCSID data conversion	
		1 Substitution characters may be used during CCSID data conversion	
		Bits 4-7: Reserved.	
141	CHAR(6)	Reserved	
147	BIN(2)	Number of devices defined for this ODP. For displays, this is determined by the number of devices defined on the DEV parameter of the Create Display File (CRTDSPF) command. For ICF, this is determined by the number of program devices defined or acquired with the Add ICF Device Entry (ADDICFDEVE) or with the Override ICF Device Entry (OVRICFDEVE) command. For all other files, it has the value of 1	All
149	CHAR(*)	Device name definition list array	All

#### MI-declaration:

```

DCL SPCPTR .OPEN-FEEDBACK;
DCL DD OPEN-FEEDBACK CHAR(256) BAS(.OPEN-FEEDBACK);
DCL DD ODP-TYPE CHAR(2) DEF(OPEN-FEEDBACK) POS( 1);
DCL DD ODP-FILE CHAR(10) DEF(OPEN-FEEDBACK) POS( 3);
DCL DD ODP-LIBRARY CHAR(10) DEF(OPEN-FEEDBACK) POS( 13);
DCL DD ODP-SPOOL-FILE CHAR(10) DEF(OPEN-FEEDBACK) POS( 23);
DCL DD ODP-SPOOL-LIB CHAR(10) DEF(OPEN-FEEDBACK) POS( 33);
DCL DD ODP-SPOOL-NBR BIN(2) DEF(OPEN-FEEDBACK) POS( 43);
DCL DD ODP-MAX-RCD-SIZE BIN(2) DEF(OPEN-FEEDBACK) POS( 45);
DCL DD ODP-MAX-KEY-SIZE BIN(2) DEF(OPEN-FEEDBACK) POS( 47);
DCL DD ODP-MEMBER CHAR(10) DEF(OPEN-FEEDBACK) POS( 49);
DCL DD * BIN(4) DEF(OPEN-FEEDBACK) POS( 59);
DCL DD * BIN(4) DEF(OPEN-FEEDBACK) POS( 63);
DCL DD ODP-DEVICE-TYPE BIN(4) DEF(OPEN-FEEDBACK) POS( 67);
DCL DD * CHAR(3) DEF(OPEN-FEEDBACK) POS( 69);
DCL DD ODP-NBR-LINES BIN(2) DEF(OPEN-FEEDBACK) POS( 72);
DCL DD ODP-NBR-COLUMNS BIN(2) DEF(OPEN-FEEDBACK) POS( 74);
DCL DD ODP-NBR-REC-OPEN BIN(4) DEF(OPEN-FEEDBACK) POS( 76);
DCL DD ODP-ACCESS-TYPE CHAR(2) DEF(OPEN-FEEDBACK) POS( 80);
DCL DD ODP-DUPL-KEY CHAR(1) DEF(OPEN-FEEDBACK) POS( 82);
DCL DD ODP-SOURCE-FILE CHAR(1) DEF(OPEN-FEEDBACK) POS( 83);
DCL DD * CHAR(10) DEF(OPEN-FEEDBACK) POS( 84);
DCL DD * CHAR(10) DEF(OPEN-FEEDBACK) POS( 94);
DCL DD ODP-VOL-LABEL BIN(2) DEF(OPEN-FEEDBACK) POS(104);
DCL DD ODP-MAX-RCD-BLK BIN(2) DEF(OPEN-FEEDBACK) POS(106);
DCL DD ODP-OVERFLOW-LN BIN(2) DEF(OPEN-FEEDBACK) POS(108);
DCL DD ODP-BLK-RCD-INC BIN(2) DEF(OPEN-FEEDBACK) POS(110);
DCL DD * BIN(4) DEF(OPEN-FEEDBACK) POS(112);
DCL DD ODP-MISC-FLAGS1 CHAR(1) DEF(OPEN-FEEDBACK) POS(116);
DCL DD ODP-REQUESTER CHAR(10) DEF(OPEN-FEEDBACK) POS(117);
DCL DD ODP-OPEN-FILES BIN(2) DEF(OPEN-FEEDBACK) POS(127);
DCL DD * BIN(2) DEF(OPEN-FEEDBACK) POS(129);
DCL DD ODP-NBR-FILES BIN(2) DEF(OPEN-FEEDBACK) POS(131);
DCL DD ODP-MISC-FLAGS2 CHAR(1) DEF(OPEN-FEEDBACK) POS(133);
DCL DD ODP-OPEN-ID CHAR(2) DEF(OPEN-FEEDBACK) POS(134);

```

```

DCL DD ODP-MAX-FMT-SIZE BIN(2) DEF(OPEN-FEEDBACK) POS(136);
DCL DD * BIN(2) DEF(OPEN-FEEDBACK) POS(138);
DCL DD ODP-MISC-FLAGS3 CHAR(1) DEF(OPEN-FEEDBACK) POS(140);
DCL DD * CHAR(6) DEF(OPEN-FEEDBACK) POS(141);
DCL DD ODP-NBR-DEVICES BIN(2) DEF(OPEN-FEEDBACK) POS(147);
DCL DD ODP-DEV-LIST... CHAR(1) DEF(OPEN-FEEDBACK) POS(149);

```

## I/O Feedback Area

The results of I/O operations are communicated to the program using OS/400 messages and I/O feedback information. The I/O feedback area is updated for every I/O operation unless your program is using blocked record I/O. In that case, the feedback area is updated only when a block of records is read or written. Some of the information reflects the last record in the block. Other information, such as the count of I/O operations, reflects the number of operations on blocks of records and not the number of records.

The I/O feedback area consists of two parts: a common area and a file-dependent area.

## Common I/O Feedback Area

The common area begins with an offset to the file-dependent area:

Position	Data Type	Contents
1	BIN(2)	Offset to file-dependent feedback area
3	BIN(4)	Write operation count. Updated only when a write operation completes successfully. For blocked record I/O operations, this count is the number of blocks, not the number of records
7	BIN(4)	Read operation count. Updated only when a read operation completes successfully. For blocked record I/O operations, this count is the number of blocks, not the number of records
11	BIN(4)	Write-read operation count. Updated only when a write-read operation completes successfully
15	BIN(4)	Other operation count. Number of successful operations other than write, read, or write-read. Updated only when the operation completes successfully. This count includes update, delete, force-end-of-data, force-end-of-volume, change-end-of-data, release record lock, and acquire/release device operations.
19	CHAR(1)	Reserved
20	CHAR(1)	Current operation
		X'01' Read or read block or read from invited devices
		X'02' Read direct
		X'03' Read by key
		X'05' Write or write block
		X'06' Write-read
		X'07' Update
		X'08' Delete
		X'09' Force-end-of-data
		X'0A' Force-end-of-volume
		X'0D' Release record lock
		X'0E' Change end-of-data
		X'0F' Put deleted record
		X'11' Release device
		X'12' Acquire device

Position	Data Type	Contents																		
21	CHAR(10)	<p>Name of the record format just processed, which is either:</p> <ul style="list-style-type: none"><li>◦ Specified on the I/O request, or</li><li>◦ Determined by default or format selection processing</li></ul> <p>For display files, the default name is either the name of the only record format in the file or the previous record format name for the record written to the display that contains input-capable fields. Because a display file may have multiple formats on the display at the same time, this format may not represent the format where the last cursor position was typed.</p> <p>For ICF files, the format name is determined by the system, based on the format selection option used.</p>																		
31	CHAR(2)	<table><tr><td colspan="2">Device class:</td></tr><tr><td>Byte 1</td><td>Byte 2</td></tr><tr><td>X'00' Database</td><td>X'00' Nonkeyed file</td></tr><tr><td></td><td>X'01' Keyed file</td></tr><tr><td>X'01' Display</td><td>X'nn'</td></tr><tr><td>X'04' Diskette</td><td>X'nn'</td></tr><tr><td>X'05' Tape</td><td>X'nn'</td></tr><tr><td>X'09' Save</td><td>X'nn'</td></tr><tr><td>X'0B' ICF</td><td>X'nn'</td></tr></table> <p>There are very many different device codes. Each time a new device has been added it has gotten its own special code</p>	Device class:		Byte 1	Byte 2	X'00' Database	X'00' Nonkeyed file		X'01' Keyed file	X'01' Display	X'nn'	X'04' Diskette	X'nn'	X'05' Tape	X'nn'	X'09' Save	X'nn'	X'0B' ICF	X'nn'
Device class:																				
Byte 1	Byte 2																			
X'00' Database	X'00' Nonkeyed file																			
	X'01' Keyed file																			
X'01' Display	X'nn'																			
X'04' Diskette	X'nn'																			
X'05' Tape	X'nn'																			
X'09' Save	X'nn'																			
X'0B' ICF	X'nn'																			
33	CHAR(10)	<p>Device name.</p> <p>The name of the device for which the operation just completed (supplied only for display, printer, tape, diskette, and ICF files).</p> <p>For printer or diskette files being spooled, the value is *N.</p> <p>For ICF files, the value is the program device name.</p> <p>For other files, the value is the device description name</p>																		
43	BIN(4)	<p>Length of the record processed by the last I/O operation (supplied only for an ICF, display, tape, or database file).</p> <p>On ICF write operations, this is the record length of the data.</p> <p>On ICF read operations, it is the record length of the record associated with the last read operation</p>																		
47	CHAR(80)	Reserved																		
127	BIN(2)	<p>Number of records retrieved on a read request for blocked records or sent on a write or force-end-of-data or force-end-of-volume request for blocked records.</p> <p>Supplied only for database, diskette, and tape files</p>																		
129	BIN(2)	<p>For output, the field value is the record format length, including first-character forms control, option indicators, source sequence numbers, and program-to-system data. If the value is zero, use the field at position 43.</p> <p>For input, the field value is the record format length, including response indicators and source sequence numbers. If the value is zero, use the field at position 43</p>																		
131	CHAR(2)	Reserved																		
133	BIN(4)	<p>Current block count. The number of blocks of the tape data file already written or read. For tape files only</p>																		
137	CHAR(8)	Reserved																		

MI-declaration:

```

DCL SPCPTR .IO-FEEDBACK;
DCL DD IO-FEEDBACK CHAR(512) BAS(.IO-FEEDBACK);
DCL DD IO-FILE-DEPNT-AREA BIN(2) DEF(IO-FEEDBACK) POS( 1);
DCL DD IO-WRITE-COUNT BIN(4) DEF(IO-FEEDBACK) POS( 3);
DCL DD IO-READ-COUNT BIN(4) DEF(IO-FEEDBACK) POS( 7);
DCL DD IO-UPDATE-COUNT BIN(4) DEF(IO-FEEDBACK) POS(11);

```



```

DCL DD IO-OTHER-COUNT      BIN(4) DEF(IO-FEEDBACK) POS(15);
DCL DD *                   CHAR(1) DEF(IO-FEEDBACK) POS(19);
DCL DD IO-CUR-OPERATION    CHAR(1) DEF(IO-FEEDBACK) POS(20);
DCL DD IO-RECORD-FORMAT    CHAR(10) DEF(IO-FEEDBACK) POS(21);
DCL DD IO-DEVICE-CLASS     CHAR(2) DEF(IO-FEEDBACK) POS(31);
DCL DD IO-DEVICE-NAME      CHAR(10) DEF(IO-FEEDBACK) POS(33);
DCL DD IO-RECORD-LENGTH    BIN(4) DEF(IO-FEEDBACK) POS(43);
DCL DD *                   CHAR(80) DEF(IO-FEEDBACK) POS(47);
DCL DD IO-RECORDS-IN-BLOCK BIN(2) DEF(IO-FEEDBACK) POS(127);
DCL DD IO-RCD-FORMAT-LENGTH BIN(2) DEF(IO-FEEDBACK) POS(129);
DCL DD *                   CHAR(2) DEF(IO-FEEDBACK) POS(131);
DCL DD IO-BLOCK-COUNT      BIN(4) DEF(IO-FEEDBACK) POS(133);
DCL DD *                   CHAR(8) DEF(IO-FEEDBACK) POS(137);

```

## I/O Feedback Area for ICF and Display Files

Position	Data Type	Contents	File Types
1	CHAR(2)	Flag bits	Display
		Bit 0: Cancel-read indicator	
		0 The cancel-read operation did not cancel the read request	
		1 The cancel-read operation canceled the read request	
		Bit 1: Data-returned indicator	
		0 The cancel-read operation did not result in a change of the contents of the input buffer	
		1 The cancel-read operation placed the data from the read-with-no-wait operation into the input buffer	
		Bit 2: Command key indicator	
		0 Conditions for setting this indicator did not occur	
		1 The Print, Help, Home, Roll Up, Roll Down, or the Clear key was pressed. The key is enabled with a DDS keyword, but without any response indicator specified	
		Bits 3-15: Reserved	
3	CHAR(1)	Attention indicator byte (AID). This field identifies which function key was pressed.  For ICF files, this field will always contain the value X'F1' to imitate the Enter key being pressed on a display device.  For display files, this field will contain the 1-byte AID value returned from the device	Display, ICF
4	CHAR(2)	Cursor location (line and position). Updated on input operations that are not subfile operations that return data to the program. For example, X'0102' means line 1, position 2. Line 10, position 33 would be X'0A21'	Display
6	BIN(4)	Actual data length. For an ICF file, see the <a href="#">ICF Programming</a> book for additional information. For a display file, this is the length of the record format processed by the I/O operation.	Display, ICF
10	BIN(2)	Relative record number of a subfile record. Updated for a subfile record operation. For input operations, updated only if data is returned to the program. If multiple subfiles are on the display, this offset will contain the relative record number for the last subfile updated	Display

Position	Data Type	Contents	File Types
12	BIN(2)	Indicates the lowest subfile relative record number currently displayed in the uppermost subfile display area if the last write operation was done to the subfile control record with SFLDSP specified. Updated for roll up and roll down operations. Reset to zero on a write operation to another record. Not set for message subfiles	Display
14	BIN(2)	Total number of records in a subfile. Updated on a put-relative operation to any subfile record. The number is set to zero on a write or write-read operation to any subfile control record with the SFLINZ keyword optioned on. If records are put to multiple subfiles on the display, this offset will contain the total number of records for all subfiles assuming that no write or write-read operations were performed to any subfile control record with the SFLINZ keyword optioned on	Display
16	CHAR(2)	Cursor location (line and position) within active window. Updated on input operations that are not subfile operations that return data to the program. For example, X'0203' means line 2, position 3 relative to the upper-left corner of the active window	Display
18	CHAR(17)	Reserved	Display, ICF
35	CHAR(2)	Major return code	Display, ICF
		"00" Operation completed successfully	
		"02" Input operation completed successfully, but job is being canceled (controlled)	
		"03" Input operation completed successfully, but no data received	
		"04" Output exception	
		"08" Device already acquired	
		"11" Read from invited devices was not successful	
		"34" Input exception	
		"80" Permanent system or file error	
		"81" Permanent session or device error	
		"82" Acquire or open operation failed	
		"83" Recoverable session or device error	
37	CHAR(2)	Minor return code (many various values)	Display, ICF
39	CHAR(8)	Systems Network Architecture (SNA) sense return code. For some return codes, this field may contain more detailed information about the reason for the error. For a description of the SNA sense codes, see the appropriate SNA manual	ICF
47	CHAR(1)	Safe indicator:	ICF
		"0" An end-of-text (ETX) control character has not been received	
		"1" An ETX control character has been received	
48	CHAR(1)	Reserved	
49	CHAR(1)	Request Write (RQSWRT) command from remote system/application	ICF
		"0" RQSWRT not received	
		"1" RQSWRT received	
50	CHAR(10)	Record format name received from the remote system	ICF

Position	Data Type	Contents	File Types
60	CHAR(4)	Reserved	
64	CHAR(8)	Mode name	ICF
72	CHAR(9)	Reserved	ICF

### I/O Feedback Area for Printer Files

Position	Data Type	Contents
1	BIN(2)	Current line number in a page
3	BIN(4)	Current page count
7	CHAR(28)	Reserved
35	CHAR(2)	Major return code
		“00” Operation completed successfully
		“80” Permanent system or file error
		“81” Permanent device error
		“82” Open operation failed
		“83” Recoverable device error occurred
37	CHAR(2)	Minor return code (many various values)

### I/O Feedback Area for Database Files

Position	Data Type	Contents
1	BIN(4)	Size of the database feedback area, including the key and the null key field byte map
5	CHAR(4)	Bits 0-31: Each bit represents a join logical file in JFILE keyword
		0 JDFTVAL not supplied for file
		1 JDFTVAL supplied for file
9	BIN(2)	Offset from the beginning of the I/O feedback area for database files to the null key field byte map which follows the key value (which begins at position 35 in this area)
11	BIN(2)	Number of locked records.
13	BIN(2)	Maximum number of fields
15	BIN(4)	Offset to the field-mapping error-bit map
19	CHAR(1)	Current file position indication
		Bit 0: Current file position is valid for get-next-key equal
		0 File position is not valid
		1 File position is valid
		Bits 1-7: Reserved
20	CHAR(1)	Current record deleted indication
		Bits 0-1: Reserved
		Bit 2: Next message indicator
		0 Next message not end of file
		1 Next message may be end of file
		Bit 3: Deleted record indicator
		0 Current file position is at an active record
		1 Current file position is at a deleted record
		Bit 4: Write operation key feedback indicator
		0 Key feedback is not provided by last write operation
		1 Key feedback is provided by last write operation

Position	Data Type	Contents
		Bit 5: File position changed indicator. Set only for read and positioning I/O operations. Not set for write, update, and delete I/O operations
		0 File position did not change
		1 File position did change
		Bit 6: Pending exception indicator. Valid for files open for input only and SEQONLY(*YES N) where N is greater than 1
		0 Pending retrieval error does not exist
		1 Pending retrieval error does exist
		Bit 7: Duplicate key indicator
		0 The key of the last read or write operation was not a duplicate key 1 The key of the last read or write operation was a duplicate key
21	BIN(2)	Number of key fields. Use this field for binary operations. Use the next field (position 22) for character operations. These fields <i>overlap</i> and provide the same value (there can be no more than 32 key fields, and only the low-order byte of position 21 is used)
22	CHAR(1)	Number of key fields
23	CHAR(4)	Reserved
27	BIN(2)	Key length
29	BIN(2)	Data member number
31	BIN(4)	Relative record number in data member
35	CHAR(*)	Key value
*	CHAR(*)	Null key field byte map

Blank