AFP Printing in an IBM Cross-System Environment

Document Number GG24-3765-00

August 1994

International Technical Support Organization
Poughkeepsie Center

Take Note!

Before using this information and the product it supports, be sure to read the general information under “Special Notices” on page ix.

First Edition (August 1994)

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

An ITSO Technical Bulletin Evaluation Form for reader's feedback appears facing Chapter 1. If the form has been removed, comments may be addressed to:

IBM Corporation, International Technical Support Organization
Dept. H52 Mailstation P099
522 South Road
Poughkeepsie, New York 12601-5400

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© **Copyright International Business Machines Corporation 1994. All rights reserved.**
Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Abstract

This document has been written for IBM customers and IBM systems engineers who are responsible for planning and installing AFP printers. It explores what needs to be done so that printing resources (fonts, segments, overlays, form definitions and page definitions) can be used on IBM AFP printers in a multiple IBM host (print anywhere from anywhere) environment. The emphasis of the book is on how to print from one system to another and how to deal with resources needed in printing. How to connect the systems is not addressed.

PR (199 pages)

© Copyright IBM Corp. 1994

Contents

© Copyright IBM Corp. 1994

Special Notices

This publication is intended for IBM customers and IBM printer specialists who are responsible for planning and installing AFP printers. The information in this publication is not intended as the specification of any programming interfaces that are provided by products mentioned in this document. See the PUBLICATIONS section of the IBM Programming Announcement for the products described in this document for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM′s product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM′s intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 208 Harbor Drive, Stamford, CT 06904 USA.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer′s ability to evaluate and integrate them into the customer′s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

The following document contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples contain the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Reference to PTF numbers that have not been released through the normal distribution process does not imply general availability. The purpose of including these reference numbers is to alert IBM customers to specific information relative to the implementation of the PTF when it becomes available to each customer according to the normal IBM PTF distribution process.

The following terms, which are denoted by an asterisk (*) in this publication, are trademarks of the International Business Machines Corporation in the United States and/or other countries:

Advanced Function Presentation	Advanced Function Printing
AFP	AIX
AIX/ESA	AIX/6000
AS/400	Bar Code Object Content Architecture

© Copyright IBM Corp. 1994

BCOCA
DisplayWrite
IBM
Intelligent Printer Data Stream
MVS/ESA
Operating System/2
OS/2
Pennant
Print Services Facility
PSF
S/370
Systems Application Architecture
VM/XA
VTAM

CICS
GDDM
IMS/ESA
IPDS
OfficeVision
Operating System/400
OS/400
Pennant Systems
PrintManager
PSF/6000
SAA
VM/ESA
VSE/ESA

Preface

This document is intended for customer and IBM printer specialists who are responsible for planning and installing AFP Printers. It explores what needs to be done to so that printing resources (fonts, segments, overlays, form definitions, and page definitions) can be used on IBM AFP printers in a multiple host (print anywhere from anywhere) environment.

The emphasis of the book is on how to print from one system to another and how to handle the resources needed to print the file correctly. How the systems are connected and how the communication parameters are set is not addressed in this document.

How This Document is Organized

The document is organized as follows:

- **Chapter 1, Introduction**

  This chapter introduces the organization of the book and provides an overview of terminology used and the basic rationale of the project used to create the book.

- **Chapter 2, Differences and Restrictions**

  This chapter describes the differences between AFP implementations in different systems and the restrictions in each implementation.

- **Chapter 3, Migrating Resources between Systems**

  This chapter has information about the AFP resources in different systems and about how to move resources between the systems.

- **Chapter 4, Printing from an MVS Host**

  This chapter discusses the considerations for submitting a file from MVS to print on AFP printers attached to other systems.

- **Chapter 5, Printing from a VM Host**

  This chapter discusses the considerations for submitting a file from VM to print on AFP printers attached to other systems.

- **Chapter 6, Printing from a VSE Host**

  This chapter discusses the considerations for submitting a file from VSE to print on AFP printers attached to other systems.

- **Chapter 7, Printing from an OS/400 Host**

  This chapter discusses the considerations for submitting a file from OS/400 to print on AFP printers attached to other systems.

- **Chapter 8, Printing from an OS/2 Host**

  This chapter discusses the considerations for submitting a file from OS/2 to print on AFP printers attached to other systems.

- **Appendix A, PSF/MVS Exits and MVS Sample Programs**

  This chapter lists the sample exits and programs written for the MVS environment.

- **Appendix B, VM AFP Sample Programs**

© Copyright IBM Corp. 1994

This chapter lists the sample exits and programs written for the VM environment.

- **Appendix C, VSE AFP Sample Programs**

  This chapter lists the sample exits and programs written for the VSE environment. There are also VSE related programs for the OS/2 environment.

- **Appendix D, OS/400 AFP Sample Programs**

  This chapter lists the programs written for the OS/400 environment, the programs run in the OS/2 environment.

Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this document.

Advanced Function Printing

- *Intelligent Printer Data Stream Reference*, S544-3417
- *Advanced Function Printing: Printer Information*, G544-3290
- *Advanced Function Printing: Data Stream Reference*, S544-3202
- *Advanced Function Printing: Host Font Data Stream Reference*, S544-3289
- *A Guide to IBM's Advanced Function Printing*, G544-3095
- *Advanced Function Printing: Printer Summary*, G544-3135
- *Advanced Function Printing: Software General Information*, G544-3415
- *AFP Resources in a Multi-System Environment*, GG24-4029

Printing in AS/400

- *Advanced Function Printing Utilities/400 User's Guide*, SH18-2416
- *AS/400 Guide to Programming for Printing*, SC41-8194
- *AS/400 Work Management Guide*, SC41-8078

Printing in OS/2

- *OS/2 Version 2.0 Technical Compendium, Volume 5: Printing Subsystem*, GG24-3775
- *Minasi, Little, Semple & Camarda: Inside OS/2 2 Special Edition, New Riders Publishing, Carmel, Indiana*

Printing in MVS

- *MVS/ESA JCL Reference*, GC28-1654
- *MVS/ESA Application Development Guide: Authorized Assembler Language Programs*, GC28-1645
- *MVS/ESA Application Development Reference: Authorized Assembler Language Programs Volume 1*, GC28-1647
- *MVS/ESA Application Development Reference: Authorized Assembler Language Programs Volume 2*, GC28-1648

- *MVS/ESA Application Development Reference: Authorized Assembler Language Programs Volume 3*, GC28-1649
- *MVS/ESA Application Development Reference: Authorized Assembler Language Programs Volume 4*, GC28-1650
- *Print Services Facility/MVS: System Programming Guide*, S544-3672
- *Print Services Facility/MVS: Application Programming Guide*, S544-3673

Printing in VM

- *VM/ESA CMS Command Reference*, SC24-5461
- *VM/ESA CP Command Reference*, SC24-5434
- *VM/ESA CP General Command Reference for 370*, SC24-5433
- *Print Services Facility/VM: System Programming Guide*, S544-3680
- *Print Services Facility/VM: Application Programming Guide*, S544-3677
- *Print Services Facility/VM: Operator's Guide*, S544-3682

Printing in VSE

- *VSE/POWER Administration and Operation*, SC33-6572
- *VSE/POWER Networking*, SC33-6573
- *VSE/POWER Application Programming*, SC33-6574
- *VSE/ESA System Control Statements*, SC33-6513
- *VSE/ESA System Macros Reference*, SC33-6516
- *Print Services Facility/VSE: System Programming Guide*, S544-3665
- *Print Services Facility/VSE: Application Programming Guide*, S544-3666

Print Services Facility/2

- *Print Services Facility/2: Getting Started*, G544-3767
- *Print Services Facility/2: Distributed Print Function Network Configuration Guide for System/370*, S544-3809
- *Print Services Facility/2: Distributed Print Function Network Configuration Guide for OS/400*, S544-3823
- *Print Services Facility/2: Type Transformer User's Guide*, G544-3796
- *PSF/2 Technical Reference*, online publication

International Technical Support Organization Publications

- *AS/400 Printing II*, GG24-3704
- *AS/400 Printing III*, GG24-4028
- *Print and View Data Streams*, GG24-3938
- *Transforming Type 1 Outline Fonts*, GG24-3964
- *AFP Resources in a Multi-system Environment*, GG24-4029

A complete list of International Technical Support Organization publications, with a brief description of each, may be found in:

Bibliography of International Technical Support Organization Technical Bulletins, GG24-3070.

To get listings of ITSO technical bulletins (redbooks) online, VNET users may type:

```
TOOLS SENDTO WTSCPOK TOOLS REDBOOKS GET REDBOOKS CATALOG
```

How to Order ITSO Technical Bulletins (Redbooks)

IBM employees in the USA may order ITSO books and CD-ROMs using PUBORDER. Customers in the USA may order by calling 1-800-879-2755 or by faxing 1-800-284-4721. Visa and Master Cards are accepted. Outside the USA, customers should contact their IBM branch office.

Customers may order hardcopy redbooks individually or in customized sets, called GBOFs, which relate to specific functions of interest. IBM employees and customers may also order redbooks in online format on CD-ROM collections, which contain the redbooks for multiple products.

Referenced Products

The following products are referenced in this document:

Program Number	Program Name	Version
OS/400		
5738-SS1	OS/400	Version 2.1
5688-179	SAA PrintManager	Release 1.0
MVS/ESA		
5695-048	MVS/ESA and JES2	Version 4.2
5695-040	PSF/MVS	Version 2.1
OS/2		
5669-336	OS/2 EE	1.3
5601-298	PSF/2 EE	1.00
VM/ESA		
5684-141	PSF/VM	Version 2.1
5684-090	RSCS	Version 3.1
5684-112	VM/ESA	Version 1.1
VSE/ESA		
5688-033	POWER	Release 5.1
5686-040	PSF/VSE	Version 2.1
5750-ACD	VSE/ESA	Release 1.2.3

Acknowledgments

The advisors for this project were:

Andy Herrup
International Technical Support Organization, Poughkeepsie Center

Mikko Markkula
International Technical Support Organization, Poughkeepsie Center

The authors of this document are:

Barry Clasper
National Support Centre IBM Canada

Therese Cowie
ISM South Africa

Geoff Kitching
IBM UK

Mikko Markkula
IBM Finland

Gert Spitzmueller
IBM Germany

Angelika Stumpf
IBM Germany

This publication is the result of residencies conducted at the International Technical Support Organization, Poughkeepsie Center.

Thanks to the following people for the invaluable advice and guidance provided in the production of this document:

Art Mazza
International Technical Support Organization, Poughkeepsie Center

Mark Dunn
International Technical Support Organization, Poughkeepsie Center

Bill Karras
International Technical Support Organization, Poughkeepsie Center

Fernando Nogal
International Technical Support Organization, Poughkeepsie Center

Helge Toftner
International Technical Support Organization, Poughkeepsie Center

Marcia Arcuri
International Technical Support Organization, Poughkeepsie Center

Michael Schwartz
International Technical Support Organization, Poughkeepsie Center

Scott Vetter
International Technical Support Organization, Poughkeepsie Center

Volker Willimzig
IBM Germany

Katherine Eland
IBM UK

Jan De Rover
IBM The Netherlands

Chapter 1. Introduction

This chapter describes the scope of this document. It also lists the test cases used in the project.

1.1 The Scope of the Document

This document is a result of various residencies and other activities at the ITSO Poughkeepsie Center.

This document concentrates on printing using Advanced Function Printing* (AFP*) in a cross-systems environment using normal transmission methods. These are methods that are included in the operating system, and mostly do not require any manual intervention or programming effort to work. There are some cases, especially printing from and to different intelligent workstations, where at least some manual actions are needed.

1.2 Methods of Communication Between Platforms

There are different ways to transmit print files between platforms:

- **Using NJE connections**

  All host systems, MVS, VM, VSE, and AS/400*, provide NJE communications. In MVS it is JES/NJE, in VM it is RSCS, in VSE it is POWER PNET, and in AS/400 the NJE communications is handled by SNADS/Bridge.

  The characteristics of a print file are included in the NJE headers. There are differences in the implementations at different platforms, so all functions available in one platform are not available in the other.

  NJE communication enables routing of print files to different systems without manual operation.

- **Using TCP/IP**

  Transmission Control Protocol/Internet Protocol (TCP/IP) includes functions to route print jobs from one system to another. There are some restrictions concerning transmission of AFP data stream files and mixed AFP data stream and line data. There are also differences in TCP/IP implementations between different platforms.

  The Line Printer Requester (LPR) command and the Line Printer Daemon (LPD) procedure handle most of the line data printing properly, but does not provide comprehensive support for AFPDS files.

- **User programming**

  It is possible to create sophisticated systems to route print jobs between platforms by writing programs. The programs and procedures may be based on APPC communication between platforms, or may be based on TCP/IP and its FTP functions.

In this document, we concentrate mainly on NJE connections, and in some cases, on manual transferring of print files from one platform to another. TCP/IP is not widely included in this document.

© Copyright IBM Corp. 1994

As PSF/6000* has no NJE connection capability, PSF/6000 and printing from and to an AIX/6000* system is not included in this document.

1.3 Test Cases

In this document, there are references to different types of test files. The following is a short description of each test case and a description of the terms used:

Filetype	Description
Flat file	A file that has no hierarchical structure. In MVS terms, a sequential data set which is comprised only of data (no print control characters). In a S/370* or OS/400* environment, these are EBCDIC and, in the OS/2* environment they are ASCII.
Line data	A flat file which is intended for printing. In many cases, there is printer control information added. This can be one or two bytes of information: • The first byte is a carriage control (CC) character which controls form movement (it can be machine code or ASA). • The second (optional) byte is the table reference character (TRC). This indicates which character set or font to use. Either parameters in JCL or installation defaults tell how to format the data. Parameters affecting the formatting are, for example, FCB, which specifies line spacing and positions on a page, where to skip when a certain CC is encountered, UCS or CHARS specifying the character set or sets used.
Line data with reference to external controls	In addition to sending line data, the user references the names of resources installed on the other system to determine how to format the data.
Line data with structured field control records	The “typical AFP application” line data is intermixed with structured fields to switch page formats or copy groups between pages, or to include images or overlays. Only a subset of the AFP data stream records are allowed mixed with line data. • Invoke Medium Map (IMM) is used to switch copy groups within a form definition. • Invoke Data Map (IDM) is used to switch page formats within a page definition. • Include Page Segment (IPS) is used to include a page segment at specified coordinates. • Include Page Overlay (IPO) is used to include a page overlay, sometimes called a “floating” overlay, at specified coordinates. • Presentation Text (PTX) objects are used to include composed page text objects. • Image objects, either IM1 or IOCA, are used to include images.

- Graphic objects are used to create graphic elements on a page.
- Bar Code objects are used to print bar codes.

In this test case, only the simple structured fields, IMM, IDM and IPO were used.

Line data mixed with complete objects

The line data is intermixed with complete AFPDS objects, like image or bar code objects.

This is in principle the same print data set type as the previous one. The only exception is that, in this case, more complex objects were also inserted in the print data.

Line data with inline PAGEDEF/FORMDEF

A resource group containing a page definition and/or form definition resource is transmitted ahead of the line data. The names of the resources must be referenced in the control information.

Line data with inline fonts

The resource group contains a complete font. This requires some awareness about the printer so that the font has the right resolution for the printer.

Except in PSF/VM, where any resource can be included to be sent with the print data set, the last three data streams require special programming. They cannot be created by all editors or by utilities.

Full AFPDS

This is the full AFP data stream such as DCF, DW/370, or OGL creates. Besides DCF, there are programs like LN2AFPDS, refer to A.6, “LN2AFPDS Program” on page 119, to convert line data into AFPDS.

With the most recent releases of Print Services Facility* program products (PSF/MVS*, PSF/VM*, PSF/VSE*, and PSF/6000*), a program called *AFP Conversion and Indexing Facility (ACIF)* was shipped. With this program it is possible to create full AFPDS files from a line data file and a page definition describing the wanted result. ACIF may be considered to be the official version of the LN2AFPDS program. ACIF supports conditional processing in page definition but LN2AFPDS does not. The LN2AFPDS program may be used with old releases of PSF where ACIF is not available, and in some cases the output of ACIF may not be accepted by that release of PSF.

With the most recent releases of Print Services Facility program products, another way to create AFPDS files was also offered. AFP Application Programming Interface (AFPAPI) provides the programmer with tools to create an AFP data stream directly from an application program (PL/1 or COBOL).

SCS data

Stands for SNA character string. It is the OS/400 print data stream, and is the OS/400 equivalent of line data.

PM metafile In OS/2 Presentation Manager*, this is a method of handling, storing and retrieving graphic representations. Metafiles do not contain pixel image, but are encoded drawing commands and coordinates that can create an image of lines, filled areas, and text strings. The pictures created are GOCA DR 3.1 and are similar to IBM* GDDM* (Graphical Data Display Manager) metafiles.

ASCII print data ASCII print data is an ASCII “flat file” that contains special printer control escape sequences within the print data. These escape sequences control actions such as line feeds, page ejects, print quality, and character set. The escape sequences are specific to a particular printer, such as a ProPrinter or a QuietWriter. Hence, you will sometimes hear such files referred to as “ProPrinter ASCII” or “QuietWriter ASCII.”

In this document, we use the expression ″worked as expected″ to say that this particular test case worked as it should work based on the information included in product manuals.

Chapter 2. Differences and Restrictions

There are some significant differences in AFP implementations in different operating systems. There are also different restrictions in each system. It is important also to emphasize that different versions of the same product may have different functions. The information included in this document refers to those program products and versions listed in “Referenced Products” on page xiv, except where clearly otherwise stated. The restrictions mentioned refer to noncustomized systems. By using user exits some restrictions can be removed. Some of these are described in the document.

2.1 Differences in AFP Implementations

All the AFP implementations have many similarities, and the basic principles are mostly the same. There are, however, some differences that are significant.

AFP implementation in OS/2 does not include an AFP resource called page definition. So in OS/2, one of the most interesting features of AFP, to be able to format a line data output file from outside of the program (so-called outboard formatting), is not possible. In OS/400, this is available when printing line data files coming from S/370 nodes.

In VM AFP, processing is split into two clearly separate phases: Spool File Conversion Machine processes the file partially and then Printer Driver Machine finally prints it. In OS/400, the implementation is similar, but the user cannot see these two phases as clearly as in VM.

2.2 Differences in Operations

There are differences in operating the AFP environments. In MVS and VSE, all the commands that are used for a non-AFP printer work with an AFP printer. These include, for example, forward spacing and backspacing. In other AFP environments, there are no commands to do this.

In MVS and VSE, it is also possible to split the output into smaller parts (segments) to allow producing the output and printing the output overlap. When segmenting output directed to an AFP printer, the user must be very careful as the segmenting originally was designed to work with non-AFP printers. The segmenting is not based on AFP pages, but rather on number of line data pages produced by the application.

2.3 Restrictions

AFP implementations in different operating environments have also different restrictions. The most significant restriction concerning page definition was already mentioned.

Another significant restriction is the maximum size of the line data record in different systems. JES2 accepts line data records up to 32 KB in length. In older JES3 releases, the limit was set by the size of buffer in the spool, as those releases did not accept line records that spanned more than one buffer. The most recent releases of JES3 (4.2.1 and later) remove this restriction. In VSE,

© Copyright IBM Corp. 1994

the maximum line data record length is 512 bytes, as this is the largest record size DTFPR accepts. It is possible in VSE to use POWER macros for writing larger records into the spool. In VSE, it is also possible to print larger line data records when the spool file is coming, for example, from MVS. In VM and VM/XA*, the size is limited to the maximum size of the virtual printer record, 204 bytes. This limitation applies also to spool files coming from other systems. When VM is used as an intermediate node between different systems, larger record sizes (up to 32KB) are accepted. VM/ESA* has a new printer type called VAFP. Using a virtual printer of this new type, it is possible to print line data records up to 32KB.

All PSF implementations do not accept the same repertoire of AFPDS records. There are more differences between different versions of a certain PSF product than between different PSF implementations. For example, the support for Graphic Objects (GOCA and BCOCA*) was not available for PSF/VSE before release 2.2. In PSF/MVS and PSF/VM, this support was included in an earlier release.

OS/400 and VM AFP implementations accept all AFP resources inline in the print data set. PSF/2 accepts all other resources inline except page definitions, which is not at all recognized by OS/2. PSF/MVS and PSF/VSE accept only form definitions and page definitions inline. With some programming work, it is possible to have a restricted support for other resources inline in print data sets printed in MVS (See Chapter 4, “Printing from an MVS Host” on page 11). With a rather recent PTF for PSF/MVS, this support for inline resources other than form definition and page definition was also provided.

2.4 PSF/2 and Distributed Print Function

Print Services Facility/2, the OS/2 implementation of Print Services Facility, is a little different from the other PSF implementation.

In addition to the normal local services available in the host implementation, PSF/2 Release 1.1 includes a special function called Distributed Print Function. This function enables using of workstation attached printers as if they were directly connected to the host system. So, the printers look quite normal system attached printers in MVS/JES, VM/CP, VSE/POWER, and OS/400. The printers are driven by the PSF/2 in the workstation. PSF/2 acting like a printer to the host, receives the print file into the IPDS part of the PSF/2 spool, and after receiving the file, PSF/2 prints the file in the order specified in the PSF/2 setup.

In a way this is cross-systems printing; but seen from the host system, it is not cross-system printing, but a printer attached in a special way.

Chapter 3. Migrating Resources between Systems

Although in most cases print files can be transferred to other nodes and printed without manual intervention, transferring resources between systems may need manual intervention.

3.1 How Resources are Stored in Different Systems

The resources are stored in a different format depending on the system. This also impacts the transferring of resources.

3.1.1 Resources in MVS

In MVS, resources are stored as members in libraries. The file format is usually variable and blocked with a machine or ASA control character (VBM or VBA). It is possible to also use fixed length records. Each resource is a separate member in the library.

Most often, resources of different types are stored in different libraries. It is common also to store IBM supplied resources and customer's production and customer's test resources in separate libraries. The libraries from which resources are fetched are specified in the start procedure of each printer.

With the latest releases of MVS, it is also possible to specify a user library in an MVS OUTPUT JCL statement to tell PSF where to find the resources for those datasets referencing to this OUTPUT statement.

In MVS, the names of a form definition or a page definition are entered in the JCL without the prefix F1 or P1. So without using PSF exits, it is only possible to have a unique name up to six characters.

3.1.2 Resources in VM

In VM, resources are stored as CMS files. The record format is usually variable, and the records include a machine or ASA control character. Each resource is a separate file on a CMS minidisk.

Different resources usually have different file types. It is common to store IBM supplied resources and customer's production and customer's test resources on different minidisks. In VM it, is possible to store the resources on a user minidisk and tell PSF either to fetch the resources from this minidisk or ask PSF to send the resources inline in the print data set. In VM, even the name of a form definition or a page definition can be unique up to eight characters.

3.1.3 Resources in VSE

In VSE, resources are stored as phases in VSE libraries. Compared with the storing format of MVS or VM, the storing format in VSE is simply described as being an object, where all the records in a resource are put after each other in sequence to form one long record. This record is then stored as a phase.

As the prefixes of resources already tell the type of the resource, there is no need to put different types of resources in separate libraries. In VSE, it is usual to have all the different resources in one library. However, most often resources

© Copyright IBM Corp. 1994

supplied by IBM are separated from the customer′s own resources, and when needed, test and production resources are stored in separate libraries.

In VSE, the names of a form definition or a page definition are entered in the POWER JECL or Printer Parameter member without the prefix F1 or P1. So it is only possible to have a unique name up to six characters.

3.1.4 Resources in OS/400

In OS/400, resources are stored as objects of special object types in OS/400 libraries. The format of resources is rather similar to that of VSE, but there are also some differences based on OS/400 library and file structure. In addition, all the objects have different type attributes in OS/400, for example, *FNTRSC refers to a font resource object, and *FORMDF refers to a form definition. It is not possible to browse or display the objects. The only way to find out the contents is to use the DMPSYSOBJ command to print the object.

As the object types of the resources already tell the type of the resource, there is no need to put different types of resources in separate libraries. In OS/400, it is usual to have all the different resources in one library. However, most often resources supplied by IBM are separated from customer′s own resources and when needed, test and production resources are stored in separate libraries.

In OS/400, the name of a form definition or a page definition can be unique up to eight characters. However, page definitions can only be referred by list files coming from S/370 environments. In MVS and VSE, it is not possible (without user modifications) to use other page definitions or form definitions than those prefixed with P1 for page definitions and F1 for form definitions. Thus, in these cases, the names can be unique only up to six characters. When the list is coming from a VM system, then the names of page definitions and form definitions can be unique up to eight characters. When form definitions are referenced locally by OS/400 applications, the name of the form definition can be unique up to eight characters as well.

A product is available in OS/400 to print files using the functions of AFP. In this application, *Advanced Function Printing Utility* (AFPU), there is an object call Printout Format Definition (PFD). Although this object has similar functions as a page definition has, it is not same as a page definition, and there is no conversion tool to convert between PFDs and page definitions.

3.1.5 Resources in OS/2

In OS/2, resources are stored as normal files on OS/2 disk. The format of resources is similar to that of VSE.

In OS/2, a resource cannot be used in PSF/2 before the resource is registered in PSF/2′s database using the RLADD command. With this command, a resource name is tied to a file name on OS/2 disk.

In OS/2, the way to separate, for example IBM supplied and customer own resources, is to define separate groups in the PSF/2 database system.

In OS/2, all the names of the resources can be unique up to eight characters. This applies to even form definitions, as there is no way to send a list file from an S/370 system to an OS/2 system so that the name of the form definition is carried over to the OS/2 system. So there are no restrictions described above in the OS/400 section.

3.2 Migrating Resources

There are different ways to move resources between systems. Some methods to do this are described below.

Resources can be moved to another system at least using the following ways:

- Manually using a tape or a diskette
- Sending resources through network as files
- Sending resources inline in a print data set and then extracting the resources in the receiving node

In the cross-system environment used in the residency producing this document, all these methods were tested and used. See the specific sections in other chapters.

3.2.1 Moving Resources Manually

Moving resources between systems can be done manually using tape or diskette as media for transportation. In some cases, some programming work is needed on both sides. When using the file transfer and a workstation, there are some special considerations to preserve the format of the resource even when transferring files between nodes of the same operating system.

3.2.2 Sending Resources through the Network as Files

Using a network in transferring resources requires that there exists a connection between the systems. If this is the case, sending resources through the network is the easiest way to handle the transmission. Because of some differences in resources in different systems, programming work may be needed.

3.2.3 Sending Resources Inline in a Print File

This method also needs connection between the nodes. In addition, some programming work is needed to extract the resources from the print file. If the resources are needed only occasionally and the receiving system accepts all needed resources inline, there is no need to copy the resources to the receiving node, but they can included inline with the print data set when needed.

Chapter 4. Printing from an MVS Host

The following chapter describes the different possibilities for AFP printing from an MVS system. It discusses the different mechanisms for requesting and transmitting a print request to a target system.

4.1 Submitting a Print Request

Since Print Services Access Facility and SAA/PrintManager have been withdrawn from marketing, there is no IBM product to offer a front-end system to submit a print job. Many customers have written their own procedures using ISPF and other functions available in the MVS system to facilitate this task.

To submit a print request in MVS, the print interface of MVS must be used. This can be done in different ways, such as submitting a utility job; or using dynamic allocation, writing the data in the spool and then using dynamic deallocation. The fact that the print file goes to a different system does not make a big difference for the keywords used for the submission. Just a nodename must be added to the printer destination. The destination information can be entered in a JES2 /*ROUTE statement, JES3 //*MAIN ORG parameter or //*FORMAT DEST parameter, or in OUTPUT JCL statement.

```
//O1 OUTPUT DEST=WTSCLSL4.PR3812
```

The previous statement specifies that the print data set referring to this OUTPUT statement will be routed to the node id WTSCSL4 and to a destination (printer) PR3812 in that node. All the AFP related parameters can be entered in the OUTPUT and DD JCL statements. This information is passed to the receiving node in NJE headers. To be sure that the resources referred to in OUTPUT statement are available in the receiving node, the resources must be migrated to the other system or, in cases when it is allowed, sent inline with the print data set.

4.1.1 Job Submission

The easiest and most used way to initiate a transmission of a print data set to another node is to submit a job that creates the spool output. Based on the information in JES2 /*ROUTE, JES3 //*MAIN or //*FORMAT statements and JCL OUTPUT and DD statements, the spool output is then transferred to another node.

```
//PRTGERTX JOB 1,'PRTGERT',MSGLEVEL=(1,1),MSGCLASS=X
//*********************************************************************
//S15  EXEC PGM=IEBGENER
//SYSPRINT  DD SYSOUT=*
//O1 OUTPUT CHARS=(GT2O,GT18,GT24,ST12),
//          DEST=WTSCSL2.PR3825,
//          FORMDEF=A1O12O,
//          PAGEDEF=W24OF3,
//          FORMS=STANDARD,
//          DATACK=UNBLOCK,
//          NOTIFY=WTSCSL2.PRTGERT,
//          NAME=('John Doe'),
//          ROOM=('1018 EG'),
//          BUILDING='03',
//          DEPT='2830',
```

© Copyright IBM Corp. 1994

```
//          ADDRESS=('Route 55','Poughkeepsie N.Y. 12602'),
//          TITLE='The pleasures of MVS JCL'
//SYSIN     DD DUMMY
//SYSUT2    DD SYSOUT=U,OUTPUT=*.O1
//SYSUT1    DD DSN=SYS1.VTAMLST(ATCCON00),DISP=SHR
//          DD DSN=SYS1.VTAMLST(NCPXPR),DISP=SHR
```

This example prints the data sets concatenated in the SYSUT1 DD statement to the print class U using the OUTPUT statement O1 in the same job step. The OUTPUT statement specifies the receiving node (WTSCSL2) and destination (PR3825) and many AFP related parameters. After the job has been written into the spool file the spool file, is transmitted by JES NJE to the receiving system.

It should be noted that some parameters are rather new and are supported only in the latest releases of JES2 (4.1.0 or later) and JES3 (4.2.1) running in MVS/ESA*. They are the parameters to facilitate the distribution of sysout data sets:

- ADDRESS
- BUILDING
- DEPT
- NAME
- ROOM
- TITLE

These parameters are supported in MVS JCL only and are not available for all other submission methods.

4.1.2 Dynamic Allocation of Sysout Data Sets

Dynamic allocation of sysout data sets with all the AFP print options is supported only as system macro. The details of the DYNALLOC macro and its parameters are described in *MVS Application Development Guide, Authorized Assembler Language Programs*.

Some subsystems provide a little simplified access to dynamic allocation. In CICS*, there are functions with which it is rather easy to specify parameters for an output file and also write the spool file itself. The following example shows how to write the SPOOLOPEN command in an CICS command level program coded in assembler language. The parameter OUTDESCR refers indirectly to a data field containing the AFP related parameters.

```
        L     R2,APARMPTR
        EXEC CICS SPOOLOPEN OUTPUT OUTDESCR(R2)
        ...
APARMPTR DC    A(PARMPTR)
PARMPTR  DC    A(PARMLEN)
PARMLEN  DC    F'44'
PARMDATA DC    CL44'DEST(PRT103) FORMDEF(CCSFDF) PAGEDEF(CCSPDF)'
```

In JES3, you cannot specify OUTPUT parameters in OUTDESCR. In JES3, it is possible to specify print class in a SPOOLOPEN command. The print class could be used to tie default AFP parameters to the output data set. This can be done in a PSF exit. A sample program to show how to set the defaults based on form name is included with PSF/MVS.

In IMS it is also possible to use dynamic allocation. This can be done directly, for example, by calling an assembler language subprogram that issues the calls for dynamic allocations. There is a so called SPE (Small Programming

Enhancement) to provide SPOOL API (Application Program Interface) services in IMS (See APARs PL72699 and PL72700).

4.1.3 Printing from the Command Level

The TSO ALLOC command does not support AFP print options like PAGEDEF and FORMDEF directly and offers limited possibilities for advanced function printing. With the ALLOC command, it is, however, possible to refer to an OUTPUT statement with an OUTDES parameter. To be able to use this method, the OUTPUT statements referred to have to be in the user′s logon procedure.

When printing from ISPF, only a batch job is submitted. There is no possibility to specify OUTPUT statements for each output data set created. By coding an OUTPUT statement with DEFAULT=YES, the resources would be used for all the print data sets produced by that particular job.

4.2 Banner Pages in MVS

Each print job can be preceded by a header separator page and followed by a trailer separator page. Different data sets of a job and copies of the same data set can be preceded by data set header pages. The information on these header, or banner pages, varies depending on where the file came from. It is possible to customize the information appearing on these pages in both JES and PSF exits.

Only the most recent versions of JES2 to and JES3 in MVS/ESA print the information of ADDRESS, BUILDING, DEPT, NAME, ROOM, TITLE on the separator page.

Depending on the system where the file came from, the information on the banner page differs.

If the file comes from a VM system, RSCS creates a jobid and a job name from an RSCS number and puts the user ID to the name-field of the banner page. Using an RSCS exit, or with a parameter in the RSCS setup, it is possible to change the information to identify the sender a little better.

The OS/400 also puts the user ID in the name field and uses a constant job name of AS400001 and a jobid of JOBnnnnn, where nnnnn is the JES-number. Except the name field we found the information rather useless.

A banner page for a file from VSE contains job name and jobid, no user ID, and in the middle of the page is block name and room from the POWER job card.

4.3 Accounting in MVS

Printing activity in MVS is recorded in SMF record type 6. In this record, there is a special section for AFP printers. For each print data set, PSF creates one record which is written before JES releases the data set to be purged. The record contains information about the job which created the data set, about the size of the data set and about the resources used to print it. It is possible to add customized information in the PSF exit.

For each data set printed, a type 6 SMF record is written independently of where the data set came from. However, none of the SMF records from outside

contained the user ID of the submitter. So it may be difficult to relate the accounting record to the submitter of the print request.

The record produced by a print job coming from VM contains the RSCS ID and the job name as RSCS creates it. Neither field is very useful for accounting.

The record caused by OS/400 is even worse. It contains the constant 'AS400001' as job name and a job number from JES2. Both VM and OS/400 manage to bring the user ID on to banner pages, but not into SMF records.

VSE does not bring the user ID into the SMF record, but the original VSE job name is preserved.

4.4 Naming of Resources

When form and page definitions are referenced in the JCL, the names are entered without the prefixes F1 and P1. So without any special processing in exits the names can be unique only up to six characters. If the CHARS parameter is used in JCL to specify a special coded font, the name of the coded font is restricted to four characters. The system will add the prefixing X0. So in this case, the resources can have a unique name up to four characters. All the other resources can, at least in principle, have a unique name up to the maximum in MVS, which is eight characters.

4.5 Printing from MVS to MVS

Printing from one MVS system to another MVS system is no more difficult than printing on only one local system. You will be using the submission methods described above and transfer the file via NJE. The format of the control language is translated by NJE to a common protocol. Even the fact that the two MVS systems may have different job entry subsystems, like JES2 and JES3, does not affect the transport and the printing on the other system.

As a NJE connection between MVS systems is common practice, we decided to have no MVS to MVS connection in our project. The observations we made from printing locally only, however, are valid for printing on two different MVS system as well.

4.5.1 Print Request Functions

This section describes the different ways to initiate a print request.

4.5.1.1 Using a Batch Job for Print Request Submission

If there is a NOTIFY parameter in the JOB card, the user gets notified via JES-Message $HASP165 that the job has completed. Since PSF/MVS Version 2, the user can have an additional NOTIFY parameter in the OUTPUT statement; then message APS063 is issued when the printing of the file is complete.

As we are dealing with systems of same kind, the information on banner pages is preserved and helps to identify the job easily.

The accounting information on the receiving side shows only the accounting data for printing.

Flat file. Worked as expected.

Line data. Worked as expected. If the DSCB of the file specifies the type of the carriage control character, this will be taken over automatically. If not, it must be specified in the JCL.

Line data referencing external resources. Worked as expected.

Line data and structured field records. Worked as expected.

Line data and image objects. Worked as expected.

Line data with inline PAGEDEF/FORMDEF. Worked as expected. It should be noted that the mere inclusion of a resource is not sufficient for the resource to be used; the resource must be externally referenced. This is done by specifying its name or DUMMY in the FORMDEF= or PAGEDEF= parameter in the OUTPUT statement. When sending a print data set to another MVS system there is no way in the JCL, to specify that certain resources are to be included, but the user has to insert them in the print file. This is very easily done, for example, using a REXX EXEC, or using the ILRPACK program described in A.5, “ILRPACK Program” on page 94.

Line data with inline fonts. In MVS, the only resources allowed in a resource group which has to precede the print data (and there can be only one resource group for a file) are FORMDEFs and PAGEDEFs. Inline fonts and overlays are not supported. However, via a record exit (APSUX04) and a resource exit (APSUX07) it is possible to extract the resource records from the data stream and put them into temporary libraries where PSF has access to them. Samples of the two exits are contained in Appendix Appendix A, “PSF/MVS Exits and MVS Sample Programs” on page 73. There is a recent official PTF to provide these functions in PSF/MVS. When sending a print data set to another MVS system, there is no way in the JCL to specify that certain resources are to be included, but the user has to insert them in the print file. This is very easily done, for example, using a REXX EXEC, or using the ILRPACK program.

Full AFPDS. Worked as expected.

The information included on the banner page includes correct information as expected.

Also the SMF records have all necessary information that is needed for accounting.

4.5.2 Resource Migration from MVS to MVS

Obviously, the migration of a resource module from one MVS system to another one is a simple operation. The formats are identical. The resources can be unloaded from a library on to a tape by using the IEBCOPY utility program and then loaded to the library of the other system using the same utility. The libraries or members of libraries can be sent to another system also by using XMIT/RECEIVE commands through a telecommunications link, if one exists between the two systems.

The steps to move resources from one MVS system to another are:

- Using telecommunications:
  - On the sending side issue the command:

    ```
    XMIT NODENAME.USER DA(RESLIB(RESNAME))
    ```

for a single resource, or

```
XMIT NODENAME.USER DA(RESLIB)
```

for a whole library.

NODENAME is the node name of the receiving node, and USER is the user to receive the transmitted resource or resources. RESLIB is the library from which the resource is sent, and RESNAME is the name of a resource in the resource library, if only one member is sent. If only one member is sent, adding parameter SEQ after causes the resource to be sent as a file containing only the resource; otherwise it would be sent as a PDS library. This is more important when sending to non-MVS systems.

- On the receiving side issue the command:

  ```
  RECEIVE
  ```

  When the system displays the name of the file to be received, issue the name of the file where you want to put the received resource. For example:

  ```
  DA(RESLIBN)
  ```

  for a library to be received, or

  ```
  DA(RESLIBN(RESNAMEN))
  ```

  for a single resource.

- Using tape:
  - On the sending side, run an IEBCOPY UNLOAD job to create a library copy on the tape, or run an IEBGENER job to create a file with the resource only.
  - On the receiving side, run either an IEBCOPY LOAD job to load the library onto a disk, or run an IEBGENER job to transfer a single resource created by an IEBGENER job.

4.6 Printing from MVS to VM

This section describes printing from MVS on a VM system.

Sending print jobs from MVS to VM is done using NJE on the MVS side and RSCS on the VM side. The systems are rather similar, and to print from an MVS system to a VM system is relatively easy.

4.6.1 Print Request Functions

This section describes the use of the different ways to initiate a print request.

4.6.1.1 Using a Batch Job for Print Request Submission

This is the conventional way to submit a print request via a batch job which executes an utility or an application program. The SYSOUT or the OUTPUT statement of the print file contains DEST=nodeid.prtid.

In addition to using normal AFP related parameters in OUTPUT statements, there is also easier ways to refer to resources when a print data set is sent from MVS to VM. It is possible in VM to set some default values for AFP parameters, such as form and page definitions, based on print class and form name.

As user notification, two messages are sent. You first receive RSCS message DMTAXM104 when the file has been written to the VM spool. This message contains only the JES2 job number. When the file has been printed, the PDM sends message APRPRT437, which contains an RSCS number and, as file name, the job name of the originating job. Especially when sending many files, it is tedious to identify which files have been printed successfully. The second message is sent only if COMPMSG is not set to NO in the PDM OPTIONS file. It is possible to get the message file produced by PSF/VM SFCM and PDM virtual machines back to the user who submitted the print job.

Job name and userid from the MVS side is shown on the banner page in the VM system, so this makes it easy to identify to whom the output belongs.

Accounting data in the receiving system shows only the data related to printing in the receiving system.

Line data. Worked as expected.

Line data referencing external resources. Worked as expected.

Line data and structured field records. Worked as expected.

Line data and image objects. Worked as expected.

Line data with inline PAGEDEF/FORMDEF. Worked as expected.

Line data with inline fonts. Worked as expected.

Full AFPDS. Worked as expected.

As VM is one of those PSF implementations accepting all resources inline in the print data set, all the test cases were printed as expected. When inline resources are included in the print data set, some programming work is needed, as MVS does not provide any tools to include them automatically. For example, a simple REXX EXEC or the ILRPACK program would do this.

4.6.2 Resource Migration from MVS to VM

Obviously, the migration of a resource module from an MVS system to a VM system is a rather simple operation. The formats are identical. The resources can be unloaded from a library on to a tape by using the IEBGENER utility program and then loaded on the disk in the VM system by using the MOVEFILE command.

MOVEFILE also allows you to receive members from a PDS library file unloaded on the MVS side by an IEBCOPY job.

The libraries or members of libraries can be sent to another system also by using a XMIT command through a telecommunications link, if one exists between the two systems. In VM, the resource is received from the reader with the RECEIVE command. If an ISPF library is sent, then ISPF is needed on the VM side as well. Sending members as sequential files does not require ISPF in VM.

The steps to migrate resources from MVS to VM are:

- Using telecommunication:
  - On the MVS side, issue the command

XMIT NODENAME.USER DA(RESLIB)

for a whole library, or

XMIT NODENAME.USER DA(RESLIB(RESNAME))

for a single member of the library. Sending a library or a member in this way requires ISPF RECEIVE for receiving on the VM side.

- A single member can be sent as a normal flat file by adding the parameter SEQ after to the XMIT command. A resource sent in this way can be received using normal RECEIVE.
- On the VM side, issue a RECEIVE command from the RDRLIST display. VM system will ask whether you want to use the name derived from the MVS side or you want to give another name for the resource on the VM side.

- Using tape:
  - On the MVS side, run either an IEBCOPY job to unload the library (or some members of a library) onto a tape, or run an IEBGENER job to create a flat file of a single member of the library.
  - On the VM side, use MOVEFILE to move the resource from the tape onto a CMS disk. If the tape file was created with IEBCOPY, the PDS parameter in MOVEFILE has to be used.

4.7 Printing from MVS to VSE

PSF implementations in MVS and VSE are rather similar. There are only some minor differences.

4.7.1 Print Request Functions

We used the standard way to submit a print request, it is the batch job. The results were identical for every case.

4.7.1.1 Using a Batch Job for Print Request Submission

When the file has arrived completely at the VSE spool, the submitter is notified with message DMTNTR147I. A print completed message is not sent.

The separator page contains the job name in block letters and a special NJE line containing the originating job number, the originating node and the originating user. If specified in the right way, even the programmer name from the account information is put on the separator page.

Accounting on the VSE system was not looked at, as VSE would have to account for the printing only.

Line data. Worked as expected.

Line data referencing external resources. Worked as expected.

Line data and structured field records. Worked as expected.

Line data and image objects. Worked as expected. This works only for image objects in PSF releases prior to 2.2. BCOCA or GOCA are supported from Release 2.2 onwards.

Line data with inline PAGEDEF/FORMDEF. Worked as expected.

Line data with inline fonts. This is not possible. VSE supports only inline PAGEDEFs and FORMDEFs.

Full AFPDS Worked as expected.

When inline resources are included in the print data set, some programming work is needed, as MVS does not provide any tools to include them automatically. For example, a simple REXX EXEC or the ILRPACK program would do this.

4.7.2 Migration of Print Resources from MVS to VSE

Besides the physical transport of a resource from MVS to VSE, it must be converted from the library format of MVS, a member of a partitioned data set, to the format of VSE. PSF/MVS provides a resource conversion utility, APTRCONV. It creates a link-edit job to be transmitted to VSE to be run there to create the phase in the VSE sublibrary.

It is possible to write the resources and link-edit job control statements on a tape and use this tape as an input for POWER. If a telecommunications link exists between MVS and VSE, then the link job can be sent directly to the POWER spool to be executed in VSE.

Steps to migrate resources from MVS to VSE:

- Using telecommunication:
  - On the MVS side run APTRCONV or a similar job to create an input stream (similar to punched cards) for the VSE linkage editor.
  - Send the created stream to VSE using the XMIT command. This can be done also by entering the proper routing information for punch data in the previous step.
  - On the VSE side, load the deck from the punch queue to an ICCF library member, check the JCL and run the linkage editor job.
- Using a tape:
  - Run the punching job (APTRCONV or similar) on the MVS side to create a tape file.
  - On VSE side start a POWER job from the tape. This will link-edit the resources to the libraries.

4.8 Printing from MVS to OS/400

The concept of NJE where there are no special transmission commands for the different file types and a printing subsystem which receives files and automatically initiates the printing is unusual for the OS/400. To avoid the manual initiation of print requests, a printer must be set up as an own userid.

OS/400 was able to handle all our requests adequately. For all our print files, there was no reason to use the manual process of RCVNETF and PRTAFPDTA.

When the RCVNETF command is used, the file has to be in fixed record format already in MVS. Even using this approach, OS/400 was able to print all the test cases correctly.

4.8.1 Print Request functions

This section describes how to initiate a print request by submitting a batch job.

4.8.1.1 Using a Batch Job for Print Request Submission

This is the conventional way to send a print file to the OS/400 and have a nodeid and a printer id in the DEST parameter of the OUTPUT or SYSOUT statement.

The originator is notified only by NJE when the file has been written to the OS/400 spool completely. There is no print complete message sent back. In case of trouble, it is up to the people at the OS/400 to dig through all the different OS/400 queues to find the reason.

One thing should be mentioned here: the banner page of the OS/400 contains very little useful information. It contains mostly fields originating from the OS/400 system itself. The only term from MVS which shows up there is the name of the job step, and it appears in the file name field.

Accounting information is not of much relevance here for two reasons: the job that created the file ran under MVS and is accounted for there, and the data available in OS/400 does not contain enough information from the sending node.

Line data. Worked as expected. ASA and machine code carriage control characters were handled correctly. Machine code characters are converted automatically.

Line data referencing external resources. Did not create a problem. Obviously, the resources referenced had to be installed on the OS/400.

Line data and structured field records. Worked as expected.

Line data and image objects. Worked as expected.

Line data with inline PAGEDEF/FORMDEF. Worked as expected. It should be mentioned that specifying FORMDEF=DUMMY to specify an inline FORMDEF does not work. In this case, OS/400 looks for a resource with the name of ′DUMMY′. Inline resources must be referenced explicitly.

Line data with inline fonts. Worked as expected.

Full AFPDS. Worked as expected.

When inline resources are included in the print data set, some programming work is needed, as MVS does not provide any tools to include them automatically. For example, a simple REXX EXEC or the ILRPACK program would do this.

4.8.1.2 Print Resource Migration

There may be cases where the receiving OS/400 system does not have all the resources required to print the job. In such cases, the resources must either be placed inline with the print file, or they must be placed in the OS/400 system′s resource libraries before the print request is submitted. Placing AFP resources inline with a print file requires user programming.

AFPDS resources can be transferred from MVS to OS/400 either using tape as the media or, if a telecommunications connection exists between MVS and

OS/400, using the XMIT command on the MVS side and the RCVNETF command in OS/400.

AFP resources have to be created with appropriate CRT commands in OS/400. These commands expect that the resources are in physical file members in OS/400. The record format in OS/400 is fixed, so to avoid problems it is best to change the record format of the resources to fixed length records already in the MVS system. In the MVS system, a file with record length and block size equal to the highest value that can be expected in the resources to be migrated and record format FB is created. The resource to be migrated is copied to this file and the file is then sent to OS/400. In the OS/400 system a physical file with the same record length is created and the transferred file is received as a member in that physical file using the RECVNETF command. Finally the resource is created using the appropriate CRT command.

Instead of using a communications line, the fixed length record file created in MVS can be moved into a tape file. This file can be copied to the physical file member in OS/400 using the CPYFRMTAP command. After that, the resource can be created using the appropriate CRT command.

The steps to move resources from an MVS system to an AS/400 system are:

- Using telecommunication:
  - On the MVS side, create a file or library with a fixed record size large enough to accommodate any resource to be migrated.
  - Copy the resource or resources to be migrated to this library with the fixed record size.
  - Send the members using the XMIT command with the SEQ parameter.
  - On the AS/400, side create a physical file with the same record size that was used in the MVS system.
  - Use the RECVNETF command to receive the sent files to this physical file.
  - Issue appropriate CRT commands to create the resources in AS/400 libraries.
- Using a tape:
  - On the MVS side, create a file or library with a fixed record size large enough to accommodate any resource to be migrated.
  - Copy the resource or resources to be migrated to this library with the fixed record size.
  - Run an IEBGENER job to move the member or members to tape file or files.
  - On the AS/400, side create a physical file with the same record size that was used in the MVS system.
  - Use the CPYFRMTAP command to receive the resources from the tape to this physical file.
  - Issue appropriate CRT commands to create the resources in AS/400 libraries.

4.9 Printing from MVS to OS/2

This section describes printing of MVS files on a printer driven by Print Services Facility* (PSF/2) on an OS/2 server.

In general, the facilities that support communications between MVS and OS2 do not facilitate printing. There is, for instance, no direct spool-to-spool communication of print files such as Network Job Entry (NJE) provides for MVS, VM, VSE, and AS/400.

Using PSF/2 Distributed Print Function, an OS/2 attached AFP printer can be used directly from the host system, but this happens using a different spool than the local PSF/2 spool.

The first requirement is a vehicle to move the print file from MVS to OS2. There are not many possibilities:

- OS/2 Communications Manager 3270 Emulation File Transfer Program

  An OS/2 user that is logged on to TSO using the 3270 Emulation capability of Communications Manager can issue an OS/2 command that will transfer a file of data from MVS to the OS/2 workstation.

- TCP/IP File Transfer Protocol

  A MVS user can use the TCP/IP FTP program to “login” to the OS/2 server and ship a file of data.

The second requirement is a mechanism to deliver the print data to the spool of the target OS/2 system, accompanied by the control information necessary to govern the printing process. Both processes described above require user interaction (that is an APRINT command) on the OS/2 machine in order to complete the printing task.

4.9.1 Technical Hurdles

There are some technical hurdles associated with printing MVS files on an OS/2 server:

- OS/2 represents data in ASCII while MVS uses EBCDIC code points. This means that normal text data must go through an EBCDIC to ASCII translation during transport. Text files that contain unusual code points may not be accurately translated. For instance, line-data files that contain machine carriage control characters may not be properly translated by the standard translate tables.
- AFPDS data, on the other hand, is exactly the same on both MVS and OS/2. That is, when transporting AFPDS data from MVS to OS/2, no conversion from EBCDIC to ASCII should be done (that is, a BINARY transfer). Therefore, it is not possible to transport files that contain both AFPDS and non-AFPDS data. The translation option chosen applies to the entire file which means it will be incorrect for part of the data.
- OS/2 has no page definition concept. The page definition is used by PSF/MVS to map line-data files to AFP pages. This mapping depends upon 1403/3800-1 line-data structures which are not native to OS/2. Therefore, PSF/2 does not employ page definitions, nor does it support MVS line-data print files, just as PSF/MVS does not support the printing of ASCII print files.

  The lack of page definition support on OS/2 has some deeper implications:

- Many of the simple page formats MVS users expect to be able to apply to print files, such as 2-up or landscape printing, are not available when the file is sent to OS/2.
- It is not possible to select a character set (font) using the CHARS parameter.
- Carriage control and TRC characters are not recognized by OS/2.
- Conditional processing is not available.

There is, however, a way to circumvent the problem of not having the page definition in the PSF/2 implementation. Using either the LN2AFPDS program listed in this document, or using AFP Conversion and Indexing Facility (ACIF) program in the MVS system, the line data file can be converted to an AFPDS file using the information in the page definition. Thus, page definition is not needed on the OS/2 side, as the file is already in AFPDS format.

4.9.2 Print Request Functions

This section describes how to receive a print data file from an MVS system to an OS/2 system and print it in the OS/2 system.

4.9.2.1 Using File Transfer to Download the Print File

If the file to be transmitted and printed is a flat file without carriage control characters and TRC characters, the file can be transmitted using EBCDIC/ASCII conversion in the file transfer. After the file has been received, it is possible to print it using the APRINT command with the appropriate parameters.

If the file is an AFPDS file, it is downloaded with the BINARY option and then printed using the APRINT command, again setting up the parameters as needed.

If the file is a mixture of line data and AFP data, or if the file is a line data file with either carriage control or TRC bytes, the file has to be converted to AFPDS to achieve a correct output. The file can be converted by using either the LN2AFPDS program or ACIF program. After conversion, the file is in AFPDS format and can be printed in the same way as any AFPDS file.

Appendix A, “PSF/MVS Exits and MVS Sample Programs” on page 73 contains the PL/I source code of an MVS program called LN2AFPDS. This program will read a line-data file, process it through a page definition, and produce an AFPDS output file. This means that all formatting normally associated with the page definition, plus the CHARS, CC, and TRC parameters are enabled. In addition, the difficulties associated with sending files that contain a mix of AFPDS and line-data are also avoided (see 4.9.1, “Technical Hurdles” on page 22).

The LN2AFPDS program does not support page definitions containing conditional processing. If conditional processing statements are encountered, LN2AFPDS ignores them.

The ACIF program shipped with the current release of PSF/MVS, supports all functions of the page definition, and can be used to transform a line data file to an AFPDS file.

As all the printing is done in the receiving system using commands in the OS/2 system, all accounting data as well as banner pages reflect the data in the OS/2 system.

4.9.3 Print Resource Migration

There may be cases where the receiving PSF/2 does not have all the resources required to print the job. In such cases, the resources must either be placed inline with the print file, or they must be placed in the PSF/2 resource libraries before the print request is submitted. The use of inline resources is enabled by the use of the LN2AFPDS utility which transforms the entire print file into AFPDS. Otherwise, the mixed data represented by a line-data file prefixed with an inline resource group would not be acceptable to PSF/2.

Creating an inline resource group usually requires some user programming. With the current level of PSF/MVS, AFP Conversion and Indexing Facility program is shipped. This program provides the necessary services to pack the resources in front of the print file. Another solution for packing the resources in front of the print file is in A.5, “ILRPACK Program” on page 94.

It is often easier to handle the requirement for a special resource by preinstalling that resource in the PSF/2 resource library.

AFPDS resources may be transferred from MVS to OS/2 using either 3270 File Transfer, or the TCP/IP File Transfer Protocol. In both cases, the transfer must be ***BINARY*** to prevent the destruction of AFPDS data. The resulting OS/2 file must then be added into the PSF/2 resource library using an RLADD command. For example, the following commands:

```
receive d:\hostfont\gx12.fnt a:'sys1.fontlibb(x0gx12)'
rladd r d:\hostfont\gx12.fnt gx12 hostres
```

would:

1. Receive member X0GX12 from MVS file SYS1.FONTLIBB through emulator session A and place it on the D drive in directory HOSTFONT as file GX12.FNT
2. Add the contents of the received file as a resource named GX12 in the PSF/2 resource group HOSTRES.

The *RECEIVE* command in the above example could have been replaced with a TCP/IP File Transfer Protocol transfer.

Chapter 5. Printing from a VM Host

The following chapter describes the different possible implementations for AFP printing from a VM system. It discusses the different mechanisms for requesting and transmitting a print request to a target system.

5.1 Submitting a Print Request

Since IBM withdrew SAA/PrintManager program product, there is no IBM product to serve as a front-end for print submission. Most of the customers have written EXECs to facilitate the submitting of print jobs.

When a user prints a file from a VM CMS virtual machine, the printout goes to the user's virtual printer. The print data set remains in the user's PRT queue unless it has not been routed to the system to be automatically printed. The user can also manually initiate the printing by changing the print data set to a certain destination and transferring it to system.

The destination of a print data set can be specified in CP SPOOL and in CP TAG DEV commands. See VM manuals for a detailed description.

```
CP SPOOL 00E DEST PR4028 CLASS A
```

specifies that all the print data sets to the virtual printer at the address 00E are routed to a destination called PR4028 and to class A.

```
CP TAG DEV 00E WTSCSL2 SYSTEM 50 SYSOUT=T
```

routes virtual printer 00E output to a node called WTSCSL2 and to the system itself. It also specifies that the priority will be 50 and the SYSOUT class will be T. This is an example of how to route the output for the virtual printer 00E to an MVS system.

To print from a VM system, there are different methods. It is possible to issue print commands from and application program and to create a spool file in this way.

5.1.1 Printing Using PRINT and PSF Commands

There are also standard commands in VM to initiate the printing of a file. The most common way to start printing of a file is to issue a PRINT command.

```
PRINT MYDATA FILE A1 (CC
```

This command causes the printing of the file MYDATA FILE A1 so that the first character in each record is interpreted to be a carriage control character (CC). The parameters that can be specified in the PRINT command do not give very many possibilities to specify the characteristics of the actual printout. For example, there are almost no AFP related parameters.

To be able to specify AFP related characteristics concerning the print data set, PSF/VM offers the PSF command. In the PSF command, there are lots of options to be specified, thus allowing the user to tell how the print file has to be printed.

```
PSF MYFILE DATA A1 (PAGEDEF(P1MYPDEF PDEF38PP A) SEND FORMDEF(F1A10111) CC
```

This command initiates printing of the file MYDATA FILE A1. The command specifies that the file contains carriage control characters (CC parameter). The

© Copyright IBM Corp. 1994

form definition to be used is called F1A10111, and it is taken from the system printing the data set. Parameter PAGEDEF says that the page definition to be used is called P1MYPDEF PDEF38PP, and it resides on the user′s A disk. Parameter SEND after the PAGEDEF tells that the page definition is sent inline with the print data set.

The size of a line data record is limited by the maximum record length of the virtual printer. In VM/SP or VM/XA, this is limited to the maximum of 204 bytes allowed by an IBM 3800 printer, in VM/ESA there is a new virtual printer type called VAFP. Using this device type, the limit is 32 KB.

When printing AFPDS documents the size is limited to 32 KB in all versions of VM. To use the PRINT command to print an AFPDS data set, parameter OVERSIZE has to be used. In addition, parameter CC must be used, as an AFPDS data set always has X′5A′ as a carriage control character in the beginning of each record.

The banner page will give the following information:

- SPOOLID - the spool file number
- CLASS
- PRINTER (destination)
- PRINT DATE
- PRINT TIME
- USER/NODEID
- FILENAME/TYPE
- FILE CREATE DATE
- FILE CREATE TIME
- DIST - Distribution information

VM collects account information about printing.

5.2 Naming of Resources

PSF/VM lets the user name the resources by using the maximum number of characters allowed by VM. The names can be unique up to eight characters. Further, the user can use a file type up to eight characters as well, giving much more freedom in naming.

5.3 Printing from VM to MVS

This section describes printing from VM on an MVS system.

Traditionally, the communication links between VM and MVS have been with RSCS and NJE being the main carriers. In addition, the software compatibilities (file structures) give no cause for cross-system print transmission concerns.

5.3.1 Print Request Functions

This section describes the use of different ways to initiate a print request.

5.3.1.1 Using Print and PSF Commands for Print Request Submission

Submitting a print job from VM to MVS is usually done by spooling the user's virtual printer to RSCS and setting the destination and other necessary parameters.

With CP TAG command it is possible to tag the spool file with information where the output for that device should go.

In our case, when a print data set was going to be sent to destination PR3825 at an MVS system called WTSCSL2, and into the SYSOUT class T, the following commands were issued.

```
CP SPOOL 00E TO RSCS DEST PR3825
CP TAG DEV 00E WTSCSL2 SYSTEM 50 SYSOUT=T
```

After these commands were issued, all output to the user's virtual printer at address 00E was directed to the correct printer.

The spool file can be written using application programs, or by using PRINT or PSF command.

PRINT command does not provide many AFP related parameters. The CC parameter is used for indicating that the first character in each record is a carriage control character. The OVERSIZE parameter is to indicate that the file is AFPDS and may have longer records than accepted as line data to the virtual printer.

If a more precise specification of AFP parameters is needed, the PSF command must be used. The PSF command allows the user to not only specify the parameters needed to make PSF/MVS print the output correctly using the resources in MVS, but also to send the resources inline with the print file. Without special programming in the MVS system, PSF/MVS does not accept any inline resources other than form and page definitions. This support for other inline resources is now available also as an official PTF for PSF/MVS.

The banner page will give the following information:

- JOB ID - MVS spool file number
- JOB NAME - The RSCS number or VM userid, selectable in RSCS customization
- SYSOUT CLASS
- DESTINATION
- NAME - userid from the VM side

Flat file. Worked as expected.

Line data. Worked as expected.

Line data referencing external resources. Worked as expected.

Line data and structured field records. Worked as expected.

Line data and image objects. Worked as expected.

Line data with inline PAGEDEF/FORMDEF Worked as expected.

Line data with inline fonts. Worked as expected.

To have inline resources other than form and page definitions accepted in a print data set sent from VM to MVS, some user programming in the MVS side is needed. In PSF/MVS the exits APSUX04 and APSUX07 were used to catch the inline resources and put them into a library in MVS and then to use those resources in printing. These functions are now shipped as an official PTF.

Full AFPDS. Worked as expected.

5.3.2 Print Resource Migration from VM to MVS

There may be cases where the receiving MVS system does not have all the resources required to print the job. In such cases, the resources must either be placed inline with the print file, or they must be placed in the MVS system′s resource libraries before the print request is submitted. Note that inline resources other than page definition and form definition are not supported unless the Inline Resource Exits described in Appendix A, “PSF/MVS Exits and MVS Sample Programs” on page 73 are installed. This support is also provided with a rather recent official PTF for PSF/MVS. Placing AFP resources inline with a print file requires user programming.

Flat files that require some special resource can only be handled by preinstalling that resource in the MVS libraries.

As the structure of the AFP resources in VM and MVS is similar, the migration of the resources is fairly simple. The resources can be moved manually using a magnetic tape as the media. The resource is written on a tape with MOVEFILE command in VM. In MVS the file can be moved on to a disk file using, for example, the IEBGENER utility program.

If there is a telecommunications connection between the systems, then the moving of resources is even easier. AFPDS may be transferred from VM to MVS using the VM SENDFILE command. The resource file may be placed in the PSF/MVS resource library using the TSO RECEIVE command.

The steps to migrate resources from a VM system to a MVS system are:

- Using telecommunication:
  - Send the resource by issuing on the VM side the SENDFILE command:

    ```
    SENDFILE RESOURCE FILE X USERID AT NODENAME
    ```

  - Receive the file issuing a RECEIVE command on the MVS side. The command will prompt for the name on the MVS side.
- Using a tape:
  - Copy the resource from the VM library to a tape file using MOVEFILE command.
  - On the MVS side, run an IEBGENER job to copy the resource from the tape file to a member in an MVS library.

5.4 Printing from VM to VM

This section describes printing from VM on a VM system.

The main ways to transmit print files between VM systems are the PRINT and PSF commands.

When printing from VM to VM, we can be talking about both from within one VM system and from a VM system to another via the conventional communication links, like RSCS.

5.4.1 Print Request Functions

This section describes how to initiate a print request.

5.4.1.1 Using Print and PSF Commands for Print Request Submission

Submitting a print job from VM to VM is usually done by spooling the user's virtual printer to RSCS with the CP SPOOL command and setting the destination and other necessary parameters.

With the CP TAG command, it is possible to tag the spool file with information where the output for that device should go.

In our case, when a print data set was going to be sent to destination 3820W10 at an VM system called WTSCPOK, and into the print class I, the following commands were issued.

```
CP SPOOL 00E TO RSCS DEST 3821W10 CLASS I
CP TAG DEV 00E WTSCPOK SYSTEM
```

After these commands were issued, all output to the user's virtual printer at address 00E was directed to the correct printer.

The spool file can be written using application programs, or by using the PRINT or PSF command.

The PRINT command does not provide many AFP related parameters. The CC parameter is used for indicating that the first character in each record is a carriage control character. The OVERSIZE parameter is used to indicate that the file is AFPDS and may have longer records than accepted as line data to the virtual printer.

If a more precise specification of AFP parameters is needed, the PSF command must be used. The PSF command allows the user to not only specify the parameters needed to make PSF/VM print the output correctly using the resources in the other VM, but also to send the resources inline with the print file. It is possible in PSF/VM to set different default values for AFP related parameters, for example, based on the print class and form name. If an agreement exists between the different systems, then the class and form name can be used to tell the receiving PSF/VM how to format the output. This is much easier than entering all the parameters in the PSF command, and this solution also works fine when using the PRINT command.

Flat file. Worked as expected.

Line data. Worked as expected.

Line data referencing external resources. Worked as expected.

Line data and structured field records. Worked as expected.

Line data and image objects. Worked as expected.

Line data with inline PAGEDEF/FORMDEF. Worked as expected.

Line data with inline fonts. Worked as expected.

Full AFPDS. Worked as expected.

Printing in the conventional manner using PSF, and PRINT commands does not pose any significant, additional or new problems.

Because there is no front-end panel to tie all the submission tasks together, each of these tasks have to be done separately and simultaneously at print time.

These tasks include:

- Spool userid printer to RSCS
- Spool prt to specific classes
- Spool prt to specific form
- Tag device printer to destination
- PSF or print file with options

The spooling of the printer commands can of course all be consolidated into one command.

A restriction of this is that each time a print file needs to be printed to a different destination or printer, these parameters have to be changed each time.

The end-user notification is standard. The following information is placed on the banner page.

- VM Spool id
- Class
- Printer id that it printed on
- System id
- Print start and finish date/time
- User/Nodeid
- File Name/Type
- File creation start and finish times
- Distribution

5.4.2 Print Resource Migration from VM to VM

There may be cases where the receiving VM system does not have all the resources required to print the job. In such cases, the resources must either be placed inline with the print file, or they must be placed in the VM system′s resource libraries before the print request is submitted. VM implementation of AFP allows any resource to be included inline in the print data set. The PSF

command provides functions to tell the sending node to include AFP resources inline with a print file.

As the structure of the resources are similar when both systems are VM systems, to move the resources is fairly simple. The resources can be moved manually using a magnetic tape as the media. The resources are written onto a tape using the TAPE DUMP command. In the receiving node, the resources are loaded onto a disk using the TAPE LOAD command.

If there is a telecommunications connection between the systems, then the moving of resources is even easier. AFPDS may be transferred from VM to VM using the VM SENDFILE command. The resource file may be placed in the other system′s library using the VM RECEIVE command.

The steps to transmit resources from one VM system to another are:

- Using telecommunications:
  - On the sending side, issue a SENDFILE command:

    ```
    SENDFILE RESOURCE FILE X USERID AT NODENAME
    ```

  - On the receiving side, issue a RECEIVE command on the RDRLIST display.
- Using tape:
  - On the sending side, dump the required resources onto a tape using TAPE DUMP command

    ```
    TAPE DUMP FN FT FM
    ```

  - On the receiving side, load the resources from the tape onto a CMS disk by using the TAPE LOAD command:

    ```
    TAPE LOAD * * FM
    ```

    .

5.5 Printing from VM to VSE

This section describes printing from VM on a VSE system.

Print files from VM to VSE are transmitted via RSCS on the VM side and POWER PNET on the VSE side.

5.5.1 Print Request Functions

This section describes how to route a print job from a VM system to a VSE system.

5.5.1.1 Using PRINT and PSF Commands for Print Request Submission

Submitting a print job from VM to VSE is usually done by spooling the user′s virtual printer to RSCS with the CP SPOOL command and setting the destination and other necessary parameters.

With the CP TAG command, it is possible to tag the spool file with information where the output for that device should go.

In our case, when a print data set was going to be sent to destination PR3816 at a VSE system called WTSCSL9, and into the SYSOUT class U, the following commands were issued.

```
CP SPOOL 00E TO RSCS DEST PR3816
CP TAG DEV 00E WTSCSL9 SYSTEM 50 SYSOUT=U
```

After these commands were issued, all output to the user's virtual printer at address 00E was directed to the correct printer.

The spool file can be written using application programs, or by using the PRINT or PSF command.

The PRINT command does not provide many AFP related parameters. The CC parameter is used for indicating that the first character in each record is a carriage control character. The OVERSIZE parameter is used to indicate that the file is AFPDS and may have longer records than accepted as line data to the virtual printer.

If a more precise specification of AFP parameters are is needed, the PSF command must be used. The PSF command allows the user to not only specify the parameters needed to make PSF/VSE print the output correctly using the resources in VSE, but also to send the resources inline with the print file. PSF/VSE does not accept any inline resources other than form and page definitions.

The banner pages provide the following information:

- Header and trailer pages
- User/printer name
- Originating userid
- Print date/time
- VSE job number

Between the two systems there is no notification of job success or failure; however there is notification that the job has arrived at VSE from VM, and this is communicated back by RSCS.

Flat file. Worked as expected.

Line data. Worked as expected.

Line data referencing external resources. Worked as expected.

Line data and structured field records. Worked as expected.

Line data and image objects. Worked as expected. The support for BCOCA and GOCA objects is included in PSF/VSE Release 2.2 and subsequent releases.

Line data with inline PAGEDEF/FORMDEF. Worked as expected.

Line data with inline fonts.

This is not possible, as PSF/VSE does not accept inline resources other than form and page definitions included in a print data set sent from VM to VSE.

Full AFPDS. Worked as expected.

5.5.2 Resource Migration from VM to VSE

There may be cases where the receiving VSE system does not have all the resources required to print the job. In such cases, the resources must either be placed inline with the print file, or they must be placed in the MVS system′s resource libraries before the print request is submitted. Note that inline resources other than page definition and form definition are not supported in VSE.

Flat files that require some special resource can only be handled by preinstalling that resource in the VSE libraries.

Any resources that are needed from the VM print file on VSE need to be transported and reformatted to be usable on VSE. A program is available on VM to do this. It creates a link-edit job that is transmitted and run on VSE to create a resource in a VSE library.

This link-edit job can be directly submitted to the VSE system′s reader if there is a communications connection between the systems. The job can be moved onto a tape and then moved manually to the VSE system to be used as input for POWER reader.

The steps to migrate resources from a VM system to a VSE system are:

- Using telecommunications:
  - In the VM system, run either an APTRCONV job or a similar job to punch the resource in a stream suitable for the VSE linkage editor.
  - Send the stream to the VSE system by using the VM PUNCH command.
  - In the VSE, load the punch file to an ICCF library member, and run the linkage editor job.
- Using a tape:
  - In the VM system, run either an APTRCONV job or a similar job to punch the resource in a stream suitable for the VSE linkage editor.
  - Copy the output of the previous step to a tape file.
  - Initiate a POWER job from the tape in the VSE system. This step link-edits the resources in VSE libraries.

5.6 Printing from VM to OS/400

This section describes printing from VM on an OS/400 system. Transfer of files along with resources is via RSCS to the OS/400. It should be noted here that each time a print file was be sent to the OS/400, a manual intervention had to be made to release the printout from the pending queue. This was generally due to the setup of the printer on the OS/400 being incompatible with the incoming forms code. Therefore, it should be recommended that the setup be made compatible with the sending forms code before print submission.

5.6.1 Print Request Functions

This section describes how to initiate a print request from a VM system to an OS/400 system.

5.6.1.1 Using Print and PSF Commands for Print Request Submission

Submitting a print job from VM to OS/400 is usually done by spooling the user's virtual printer to RSCS and setting the destination and other necessary parameters.

With the CP TAG command, it is possible to tag the spool file with information where the output for that device should go.

In our case, when a print data set was going to be sent to destination PR3812 at an OS/400 system called WTSCSL4, and into the SYSOUT class A, the following commands were issued.

```
CP SPOOL 00E TO RSCS DEST PR3812 CLASS A
CP TAG DEV 00E WTSCSL4
```

After these commands were issued, all output to the user's virtual printer at address 00E was directed to the correct printer.

The spool file can be written using application programs, or by using the PRINT or PSF command.

The PRINT command does not provide many AFP related parameters. The CC parameter is used for indicating that the first character in each record is a carriage control character. The OVERSIZE parameter is used to indicate that the file is AFPDS and may have longer records than accepted as line data to the virtual printer.

If a more precise specification of AFP parameters are is needed, the PSF command must be used. The PSF command allows the user to not only specify the parameters needed to make OS/400 print the output correctly using the resources in OS/400, but also to send the resources inline with the print file. OS/400 accepts any resources inline with the print data set.

The banner page in an OS/400 system does not include very much data from the submitting system, as OS/400 prints data related to its internal print job on the page.

Flat file. Worked as expected.

Line data. Worked as expected.

Line data referencing external resources. Worked as expected.

Line data and structured field records. Worked as expected.

Line data and image objects. Worked as expected.

Line data with inline PAGEDEF/FORMDEF. Worked as expected.

Line data with inline fonts. Worked as expected.

Full AFPDS. Worked as expected.

5.6.2 Print Resource Migration from VM to OS/400

There may be cases where the receiving OS/400 system does not have all the resources required to print the job. In such cases, the resources must either be placed inline with the print file, or they must be placed in the OS/400 system′s resource libraries before the print request is submitted.

Flat files that require some special resource can only be handled by preinstalling that resource in the OS/400 libraries.

If the resources are needed occasionally, they can be sent inline with the print data set. PSF/VM provides tools to include the resources, and OS/400 accepts any resource inline.

The resources can migrated from VM to OS/400 manually using magnetic tape. If a communications connection exists between the systems, the task is easier.

In both cases, a fixed length record has to be created in the VM system. The record length must be set to the maximum value that can be expected in the resources concerned. The file can be very easily created with the COPY command in VM. The original file is copied to a temporary member with specifying in COPY options (RECFM F LRECL xxxx, where xxxx is the maximum record length needed. The file is then either moved with MOVEFILE to a tape file and moved manually to the OS/400 system, or sent through the communications link to OS/400. In OS/400, the file is either copied with CPYFRMTAP from a tape onto a disk, or received with RCVNETF to a physical file member. The physical file should have a record length that is at least as long as the resources received.

With CRT commands, the resources are created using the received physical file members as input files.

The steps to migrate resources from a VM system to an AS/400 system are:

- Using telecommunication:
  - On the VM side, copy the resource to a file with fixed record large enough to accommodate any record in the resource to be migrated.

    ```
    COPY RESOURCE FILE X RESOURCE FIXFILE X (RECFM F LRECL maxl
    ```

  - Send the files using the SENDFILE command.
  - On the AS/400, side create a physical file with the same record size that was used in the MVS system.
  - Use the RCVNETF command to receive the sent files to this physical file.
  - Issue appropriate CRT commands to create the resources in AS/400 libraries.
- Using a tape:
  - On the VM side, copy the resource to a file with a fixed record size large enough to accomodate any record of the resource to be migrated.

    ```
    Copy RESOURCE FILE X RESOURCE FIXFILE X (RECFM F LRECL maxl
    ```

  - Use the MOVEFILE command to copy the file to a tape file.
  - On the AS/400 side, create a physical file with the same record size that was used in the MVS system.

- Use the CPYFRMTAP command to receive the resources from the tape to this physical file.
- Issue appropriate CRT commands to create the resources in AS/400 libraries.

5.7 Printing from VM to OS/2

This section describes printing of VM files on a printer driven by PSF/2 on an OS/2 server.

In general, the facilities that support communications between VM and OS2 do not facilitate printing. There is, for instance, no direct spool-to-spool communication of print files, such as Network Job Entry (NJE) provides for MVS, VM, VSE, and AS/400.

The Distributed Print Function available in PSF/2 Release 1.10 provides a way to use OS/2 workstation attached AFP supported printers directly from the VM system. This is not actually printing from a VM system to an OS/2 system, although the printer driver resides in the OS/2 PSF/2 program. From the VM system, the printer looks like any other AFP printer attached to the VM system. PSF/2 acts as a printer to the VM system, receives the print file to the IPDS part of the PSF/2 spool, and prints the file when it is its turn to be printed.

Otherwise, the printing from a VM system to the OS/2 system requires manual intervention.

The first requirement is a vehicle to move the print file from VM to OS2. There are not many possibilities:

- OS/2 Communications Manager 3270 Emulation File Transfer Program

  An OS/2 user that is logged on to CMS using the 3270 Emulation capability of Communications Manager can issue an OS/2 command that will transfer a file of data from VM to the OS/2 workstation.

- TCP/IP File Transfer Protocol

  A VM user can use the TCP/IP FTP program to “login” to the OS/2 server and ship a file of data.

The second requirement is a mechanism to deliver the print data to the spool of the target OS/2 system, accompanied by the control information necessary to govern the printing process. Both processes described above require user interaction (it is an APRINT command) on the OS/2 machine in order to complete the printing task.

5.7.1 Technical Hurdles

There are some technical hurdles associated with printing VM files on an OS/2 server:

- OS/2 represents data in ASCII while VM uses EBCDIC code points. This means that normal text data must go through an EBCDIC to ASCII translation during transport. Text files that contain unusual code points may not be accurately translated. For instance, line-data files that contain machine carriage control characters may not be properly translated by the standard translate tables.

- AFPDS data, on the other hand, is exactly the same on both VM and OS/2. That is, when transporting AFPDS data from VM to OS/2, no conversion from EBCDIC to ASCII should be done (that is BINARY transfer). Therefore, it is not possible to transport files that contain both AFPDS and non-AFPDS data. The translation option chosen applies to the entire file which means it will be incorrect for part of the data.
- OS/2 has no page definition concept. The page definition is used by PSF/VM to map line-data files to AFP pages. This mapping depends upon 1403/3800-1 line-data structures which are not native to OS/2. Therefore, PSF/2 does not employ page definitons, nor does it support VM line-data print files, just as PSF/VM does not support the printing of ASCII print files.

  The lack of page definition support on OS/2 has some deeper implications:

  – Many of the simple page formats VM users expect to be able to apply to print files, such as 2-up or landscape printing, are not available when the file is sent to OS/2.

  – It is not possible to select a character set (font) using the CHARS parameter.

  – Carriage control and TRC characters are not recognized by OS/2.

  – Conditional processing is not available.

  There is, however, a way to circumvent the problem of not having the page definition in the PSF/2 implementation. Using either the LN2AFPDS program (see A.6, “LN2AFPDS Program” on page 119) listed in this document, or using the AFP Conversion and Indexing Facility (ACIF) program in the VM system, the line data file can be converted to an AFPDS file using the information in the page definition. The LN2AFPDS program does not support conditional processing in the page definition, the ACIF program has this support included. Thus, page definition is not needed on the OS/2 side, as the file is already in AFPDS format.

5.7.2 Print Request Functions

This section describes how to receive a print data file from an MVS system to an OS/2 system and print it in the OS/2 system.

5.7.2.1 Using File Transfer to Download the Print File

If the file to be transmitted and printed is a flat file without carriage control characters and TRC characters, the file can be transmitted using EBCDIC/ASCII conversion in the file transfer. After the file has been received, it is possible to print it using the APRINT command with the appropriate parameters.

If the file is an AFPDS file, it is downloaded with the BINARY option and then printed using the APRINT command, again setting up the parameters as needed.

If the file is a mixture of line data and AFP data, or if the file is a line data file with either carriage control or TRC bytes, the file has to be converted to AFPDS to achieve a correct output. The file can be converted by using either the LN2AFPDS program or ACIF program. After conversion, the file is in AFPDS format, and can be printed in the same way as any AFPDS file.

5.7.3 Print Resource Migration from VM to OS/2

There may be cases where the receiving PSF/2 does not have all the resources required to print the job. In such cases, the resources must either be placed inline with the print file, or they must be placed in the PSF/2 resource libraries before the print request is submitted. The use of inline resources is enabled by the use of the LN2AFPDS utility which transforms the entire print file into AFPDS. With the current level of PSF/VM, the AFP Converison and Indexing Facility (ACIF) program is shipped. This program includes the necessary services to pack the resources in front of the print file. Otherwise, the mixed data represented by a line-data file prefixed with an inline resource group would not be acceptable to PSF/2.

It is often easier, therefore, to handle the requirement for a special resource by preinstalling that resource in the PSF/2 resource library.

AFPDS resources may be transferred from MVS to OS/2 using either 3270 File Transfer, or the TCP/IP File Transfer Protocol. In both cases, the transfer must be ***BINARY*** to prevent the destruction of AFPDS data. The resulting OS/2 file must then be added into the PSF/2 resource library using an RLADD command. For example, the following commands:

```
receive d:\hostfont\gx12.fnt a:x0gx12 font3820 m
rladd r d:\hostfont\gx12.fnt gx12 hostres
```

would:

1. receive VM file X0GX12 FONT3820 M through emulator session A and place it on the D drive in directory HOSTFONT as file GX12.FNT
2. Add the contents of the received file as a resource named GX12 in the PSF/2 resource group HOSTRES.

The *RECEIVE* command in the above example could have been replaced with a TCP/IP FTP transfer.

Chapter 6. Printing from a VSE Host

The following chapter describes the different possible implementations for AFP printing from a VSE system. It discusses the different mechanisms for requesting and transmitting a print request to a target system.

6.1 AFP Printing in VSE

Although VSE belongs to the S/390 family, VSE is a little different from MVS and VM environments. The product repertoire to transmit print requests and objects for printing is much less than in other S/390 systems. For example, transmitting AFP resources is not possible in the same way as it is in VM.

VSE does not provide such printing utilities as PRINT or PSF commands in VM, or IEBGENER in MVS.

When receiving AFP print jobs from other nodes PSF/VSE accepts only form definitions and page definitions as inline resources. We were not able to circumvent this restriction of PSF/VSE, as there are no user exits to intercept the printing in the same way as it is possible in PSF/MVS.

Even sending form definitions and page definitions as an inline resource requires programming.

To be able to transmit any of the resources in front of the print data set, a program was written for this purpose (C.4, “Program to Punch an AFP Resource Inline” on page 169).

VSE/POWER inserts a control record with X′73′ as the carriage control byte in front of the print data set, when the print data set is transmitted to another node. When inline resources are included in front of the print data set, this control record has to be removed, as the receiving system (if it is not another VSE/POWER) does not accept any records in front of the inline resource group.

PSF/VSE supports AFP data stream. GOCA and BCOCA are supported from Release 2.2 onwards.

To have the capability to communicate with other systems, POWER has to be generated with PNET support. POWER node table has to be customized to include all the nodes with which communication is needed.

There are two ways to specify the AFP formatting resources (PAGEDEF and FORMDEF) in VSE. The user can include SET commands in the POWER AUTOSTART deck to enable the use of the AFP related keywords in the POWER JECL LST statement. The user can also use a way that is more compatible with line printing with preprinted forms. By creating a special object called printer parameter member, a user can include references to the AFP resources, such as FORMDEF, PAGEDEF and CHARS. This printer parameter member can be referred to in the JECL LST statement keyword FNO. Thus replacing an old preprinted form with AFP resources does not need any change in the JCL or POWER JECL.

© Copyright IBM Corp. 1994

From VSE, all the print requests to MVS, VM and AS/400 can be routed by coding the DEST parameter in the POWER JECL LST statement, or by coding the LDEST parameter in the JECL JOB statement.

There is no standard method to route print data sets to OS/2.

PSF/2 Release 1.10 includes the Distributed Print Function that provides a way to use an OS/2 attached AFP printer to act like any AFP printer in the VSE system. PSF/2 emulates an AFP printer to the VSE system, receives the print data into the PSF/2 spool, and finally prints it in the order set in the system setup for PSF/2. This is not cross-system printing, but a printer attached to the host in a special way.

The DEST parameter specifies receiving node identification and when needed, also the user identification in the receiving node. The user identification usually indicates a specific printer. For example, coding DEST=(WTSCSL2) tells VSE to route the output to the destination WTSCSL2; coding DEST=(WTSCSL2,PR3825) tells VSE to route the output to the printer PR3825 in the node WTSCSL2.

If you want to have banner pages, or separator pages as they a called in VSE, you specify it in POWER generation. The values specified in generation can be overridden with a POWER JECL LST statement.

There is no exit in PSF/VSE to customize the separator pages. Only functions available in form definitions and page definitions can be used for customizing.

6.2 Printing from VSE to MVS

This section describes printing from VSE on an MVS system.

6.2.1 Print Request Functions

This section describes how to use the VSE POWER JECL to initiate a print request. The following types of data are considered:

6.2.1.1 Using POWER JECL for Print Request Submission

All the different file types are handled in the same way in POWER JECL.

Some of the information included in the POWER JOB statement is transmitted to the receiving node and included in the banner page.

If there are no inline resources included, then only the appropriate keywords in POWER JECL JOB or LST statements have to be coded. These keywords are LDEST in JOB statement or DEST in LST statement to route the output and the CLASS parameter to select the print class. The AFP related parameters can be included in a printer parameter member or specified as keywords in the POWER JECL LST statement. The printer parameter member is referenced in the LST statement by coding the FNO keyword. To be able to use the AFP keywords, such as FORMDEF, PAGEDEF and so on, the POWER AUTOSTART deck has to include SET commands for these keywords.

It is possible to get a message notifying you that the output has been routed to another node, but it is not possible to get a notification indicating that the printing has finished.

All the subtypes of print data sets mentioned in the table are handled in the same way.

6.2.2 Print Resource Migration

In many instances, you will want to migrate print resources from the VSE system to the MVS platform. To migrate print resources from VSE to MVS, some programming work is needed, as the file transfer functions available in VSE do not provide functions for transmitting AFP resources.

To ″punch″ an AFP resource onto cards, a program (C.1, “Program to Punch an AFP Resource for MVS” on page 163) was written. Then JCL and the punched output was sent to MVS. In MVS, another program (C.3, “Program to Create a Resource from VSE Punch Output” on page 168) was written to create the resource from the input.

The procedure described above is needed if the resources are moved through a network using telecommunications.

If a tape file is used, the resource can be copied as a flat, variable-length record file to a tape and then loaded from the tape to an MVS library. A program to move a resource onto a tape is described in C.6, “Program to Create a Tape File for MVS or VM” on page 175.

The steps to migrate resources from a VSE system to an MVS system are:

- Using telecommunications:
  - Use a program in the VSE system to ″punch″ the resource into the POWER queue. With proper parameters, the job is automatically sent to the MVS system.
  - If the file is not automatically routed to the MVS system, it can be loaded to an ICCF library and the sent to an MVS user ID.
  - On the MVS side, in the case where the job is not automatically started from the VSE system, the file can be received, and then the job can be run in MVS to create the resource in an MVS library.
- Using a tape:
  - In the VSE system, the resource is copied to a tape file, for example, using the program described in C.6, “Program to Create a Tape File for MVS or VM” on page 175.
  - In the MVS side, an IEBGENER job is run to load the resource to an MVS system.

6.3 Printing from VSE to VM

This section describes printing from VSE on an VM system.

6.3.1 Print Request Functions

This section describes how to use VSE POWER JECL to initiate a print request.

6.3.1.1 Using POWER JECL for Print Request Submission

All the different file types are handled in the same way in POWER JECL.

Some of the information included in the POWER JOB statement is transmitted to the receiving node and included in the banner page.

If there are no inline resources included, then only the appropriate keywords in POWER JECL JOB or LST statements have to be coded. These keywords are LDEST in JOB statement or DEST in LST statement to route the output and the CLASS parameter to select the print class. The AFP related parameters can be included in a printer parameter member or specified as keywords in the POWER JECL LST statement. The printer parameter member is referenced in the LST statement by coding the FNO keyword. To be able to use the AFP keywords, such as FORMDEF, PAGEDEF and so on, the POWER AUTOSTART deck has to include SET commands for these keywords.

It is possible to get a message notifying you that the output has been routed to another node, but it is not possible to get a notification indicating that the printing has finished.

All the subtypes of print data sets mentioned in the table are handled in the same way.

6.3.2 Print Resource Migration

In many instances, you will want to migrate print resources from the VSE system to the VM platform. To migrate print resources from VSE to VM, some programming work is needed, as the file transfer functions available in VSE do not provide functions for transmitting AFP resources.

A program to ″punch″ a resource to an entry in the POWER queue is described in C.2, “Program to Punch an AFP Resource for VM” on page 166. This program creates a file that can be loaded to an ICCF library member and then sent to the VM system. By setting the the PDEST parameter in the POWER JECL JOB statement correctly, it is possible to route the the puched output directly to a user ID in the VM system. On the VM side, an EXEC to create the resource from this file is used. See C.5, “Program to Create a Resource in VM” on page 174.

It is also possible to use a program to copy the resource to a tape file with variable length records. This file can then be copied to the VM system with the MOVEFILE command.

The steps to migrate a resource from a VSE system to a VM system are:

- Using telecommunications:
  - Use a program in the VSE system to ″punch″ the resource into the POWER queue. With proper parameters, the file is automatically sent to the VM system.
  - If the file is not automatically routed to the VM system, it can be loaded to an ICCF library and the sent to an VM user ID.
  - On the VM side, the file can be received, and then an EXEC can be run in VM to create the resource on a CMS disk.
- Using a tape:

- In the VSE system the resource is copied to a tape file, for example, using the program described in C.6, “Program to Create a Tape File for MVS or VM” on page 175.
- On the VM side, the file can be copied onto a CMS disk by using the MOVEFILE command.

6.4 Printing from VSE to VSE

This section describes printing from VSE on a VSE system.

If the VSE systems use a shared spool, no sending of the print data set is needed. Both systems can write files into the spool and both systems can print files from the spool onto the printers.

6.4.1 Print Request Functions

This section describes the use of the VSE POWER JECL to initiate a print request.

6.4.1.1 Using POWER JECL for Print Request Submission

All the different file types are handled in the same way in POWER JECL.

Some of the information included in POWER JOB statement is transmitted to the receiving node and included in the banner page.

If there are no inline resources included, then only the appropriate keywords in POWER JECL JOB or LST statements have to be coded. These keywords are LDEST in JOB statement or DEST in the LST statement to route the output and the CLASS parameter to select the print class. The AFP related parameters can be included in a printer parameter member or specified as keywords in the POWER JECL LST statement. The printer parameter member is referenced in the LST statement by coding the FNO keyword. To be able to use the AFP keywords, such as FORMDEF, PAGEDEF and so on, the POWER AUTOSTART deck has to include SET commands for these keywords.

It is possible to get a message notifying you that the output has been routed to another node, but it is not possible to get a notification indicating that the printing has finished.

All the subtypes of print data sets mentioned in the table are handled in the same way.

6.4.2 Print Resource Migration

In many instances, you will want to migrate print resources from the VSE system to another VSE platform. To migrate print resource, from VSE to another VSE system is possible by using the standard functions of VSE.

As the systems are similar, it is easy to migrate resources from one VSE system to another. The easiest way is to use VSE Librarian program to copy the resources from the sending system to a tape, and then in the receiving system again by using the Librarian program restore the resources to the receiving system′s library.

If the user wants to use direct communications between systems, it is also possible to move resources from one system to another. The resources can be

″punched″ in the sending system by using the Librarian program and then loaded to an ICCF library. The members in the ICCF library can be sent to the other VSE system by using functions included in the VSE system.

6.5 Printing from VSE to AS/400

This section describes printing from VSE on an AS/400 system.

Officially, a direct NJE connection from VSE to AS/400 is not supported, although it has been successfully tested. In our case, we used VM and RSCS as an intermediate node.

6.5.1 Print Request Functions

This section describes the use of the VSE POWER JECL to initiate a print request.

6.5.1.1 Using POWER JECL for Print Request Submission

All the different file types are handled in the same way in POWER JECL.

No information included in POWER JOB statement is transmitted to the receiving node and included in the banner page.

If there are no inline resources included, then only the appropriate keywords in POWER JECL JOB or LST statements have to be coded. These keywords are LDEST in JOB statement or DEST in LST statement to route the output, and CLASS parameter to select the print class. The AFP related parameters can be included in a printer parameter member or specified as keywords in the POWER JECL LST statement. The printer parameter member is referenced in the LST statement by coding the FNO keyword. To be able to use the AFP keywords, such as FORMDEF, PAGEDEF and so on, the POWER AUTOSTART deck has to include SET commands for these keywords.

It is possible to get a message notifying you that the output has been routed to another node, but it is not possible to get a notification indicating that the printing has finished.

All the subtypes of print data sets mentioned in the table are handled in the same way.

6.5.2 Print Resource Migration

In many instances, you will want to migrate print resources from the VSE system to the AS/400 platform. To migrate print resources from VSE to AS/400, some programming work is needed, as the file transfer functions available in VSE do not provide functions for transmitting AFP resources.

If you want to use telecommunications, the resource to be migrated has to be converted to a format that is suitable for transmission. One way to do this is to use a program, for example C.2, “Program to Punch an AFP Resource for VM” on page 166, to ″punch″ the resource into an entry in the POWER/VSE queue, then load this punch file to an ICCF library member, and then finally send it to the AS/400 system. A program is needed in AS/400 to create a physical file member from the received file. From the physical file member it is possible to create an AFP resource in the AS/400 system library by using appropriate CRT commands.

Using tape, the procedure is a little easier. For example, by using a program described in C.7, “Program to Create a Tape File for AS/400” on page 176, it is possible to copy a resource from a VSE library to a tape file with fixed record length. In AS/400, the file can be copied to a physical file member by using the CPYFRMTAP command. And finally, the resource can be created from the physical file member by using an appropriate CRT command.

6.6 Printing from VSE to OS/2

This section describes printing from VSE on an OS/2 system.

There is no direct way to submit a print job to an OS/2 system. Programming work is needed to provide this capability.

With PSF/2 Release 1.10, it is possible to use Distributed Print Function for printing. In this case, PSF/2 acts like a printer to the VSE system, receives the print file into the IPDS spool of PSF/2, and finally prints it on a printer attached to the PSF/2. This is not actually cross-system printing, but a different way to attach a printer to the VSE system,

It is possible to manually transfer print files from the VSE/POWER queue to a workstation and then print the file in the OS/2 workstation. This works rather well with flat files without any carriage control characters or TRC characters, but does not work with mixed AFPDS and line data or line data with control characters.

It would be possible - with rather much programming work - to have a sort of automatic print submission, for example using APPC in VSE/CICS and the OS/2 workstation. In the time frame for the residency, however, this was not even tried.

6.6.1 Print Resource Migration

Moving resources from a VSE system to an OS/2 system needs quite a lot of programming. There are some examples of programs showing how to migrate resources from a VSE system to an OS/2 system in Appendix C, “VSE AFP Sample Programs” on page 163.

Chapter 7. Printing from an OS/400 Host

The following chapter describes the different possible implementations for AFP printing from an OS/400 system. It discusses the different mechanisms for requesting and transmitting a print request to a target system.

As there is an excellent document on AS/400 Printing (*IBM AS/400 Printing III*) available, there is no need to go into very small details in this document.

Compared with the other environments, the capabilities to create print files with different characteristics are fewer. For example, including resources inline means a lot of programming work either using SAA PrintManager or other methods. The resource objects in OS/400 are inaccessible by normal programming tools. Using Machine Interface (MI) and C/400 with PRPQ P10102, it may be possible to access these objects and even create a complex print data set with mixed line and AFP data. In the time frame available, it was impossible to accomplish any tests using these methods.

PSF was included in OS/400 as a part of the operating system, so in this way it is different from the other platforms where PSF is available. In the most recent releases, PSF/400 is a separately orderable program product.

AFP implementation in OS/400 does not include page definitions for print files that are produced with OS/400 applications. Page definitions are included in the implementation only to enable printing of line data coming from S/370 nodes.

PSF in OS/400 is similar to the implementation in VM as inline resources are concerned. PSF in OS/400 accepts any AFP resources included in the print data stream. It is possible to create a spool file in the OS/400 system with an application program or using the OS/400 services from the panels.

OS/400 produces header pages with some useful information only if the print file is originating from the printing system itself. When a print file received from another node is printed, hardly any information related to the originating node can be found.

OS/400 also provides functions for collecting account information. This is described in the *Work Management Guide* manual. The accounting information only pertains to the operations in the printing node, and as no information coming from the other nodes is included, it is not possible to use the data gathered by OS/400 to, for example, charging the remote users for services.

There are some products that produce AFPDS output in OS/400.

Advanced Function Printing Utilities (AFPU) provide a method to create formatting objects called Printout Format Definition (PFD). These objects can be used as formatting resources by AFPU when printing a database file. This method resembles using page definitions in S/370 systems. PFD has some functions that are not included in page definitions and vice versa. The resource PFD and page definition are not similar. No tools to convert these objects to each other exist. AFPU has a function to print an overlay or an page segment. This function creates a spool file with the resource as an inline resource followed by an AFP document referring to this object. Because of that, the receiving system has to have the capability to accept overlays and page

© Copyright IBM Corp. 1994

segments as inline resources if the output of the program is sent to another node.

7.1 Print Request Functions

This section describes the ways to initiate a print request from an OS/400 system.

7.1.1 Using Send Network Spooled File (SNDNETSPLF) Command

No matter how the spool file is created, it can be sent to another node after the spool file is in the OS/400 spool.

Sending a print file from an OS/400 system to another system is initiated by using the Send Network Spooled File (SNDNETSPLF) command. This command can be entered on the command line in an OS/400 system, or it can included in a program. It is also possible to initiate the sending of a spool file when working with output queues (WRKOUTQ). It is possible to enter code indicating a request to send the file on one of the screens. In each case, the SNDNETSPLF command is issued as the final step.

The use of the SNDNETSPLF command does not give a large repertoire of parameters to be attached to the spool file sent. It is possible, of course, to give the destination and node ID of the printer in the receiving node. It is also possible to specify some other parameters, for example output class. AFP related parameters, such as form definition, page definition, or characters to be used cannot be entered in the SNDNETSPLF command.

Using the Work with Output Queue (WRKOUTQ) command it is possible to change, for example, the form name, but there is no way to specify, for example, the name of the form definition so that information would be passed to the receiving system.

The menu screen of the SNDNETSPLF command is shown below.

```
                    Send Network Spooled File (SNDNETSPLF)
Type choices, press Enter.
Spooled file . . . . . > QSYSPRT       Name
User ID:
  User ID  . . . . . .   PR3825        Character value
  Address  . . . . . .   WTSCSL2       Character value
              + for more values
Job name . . . . . . . > DSP11         Name, *
  User . . . . . . . . >   PRTANGEL    Name
  Number . . . . . . . >   008073      000000-999999
Spooled file number  . . . . . > 11    1-9999, *ONLY, *LAST
Data format  . . . . . .   *RCDDATA    *RCDDATA, *ALLDATA
                         Additional Parameters
VM/MVS class . . . . . .   A           A, B, C, D, E, F, G,  , I...
                                                                Bottom
F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
F24=More keys
```

The ″User ID″ (PR3825) specifies the destination in the in the receiving node (WTSCSL2) that is specified in the field ″Address″. Print class can be specified in ″VM/MVS class″. This works also with VSE.

There are two possibilities to specify the ″Data format″, either *RCDDATA or *ALLDATA. When printing print data sets that where originally designed to be printed on an SCS printer using *RCDDATA transfers the print file itself, but the characteristics of the print file, such as drawer and form type, are not transferred. By specifying *ALLDATA, all related information is also passed to the receiving node. When a print data set including AFPDS or AFPDS and line data is sent to another node, *ALLDATA must be specified.

In any case, there are restrictions in passing parameters to other nodes. So without any programming effort on either the OS/400 side or in the receiving node, print files are printed using the default value in the receiving node.

7.1.2 Printing Using SAA PrintManager

SAA PrintManager provides the user with more possibilities than SNDNETSPLF to enter information about the characteristics of the print file.

The only way to use SAA PrintManager from OS/400 is to use an application program written in C, COBOL, or RPG. Using SAA PrintManager, it is possible to specify most of the AFP related parameters.

7.2 Printing from OS/400 to MVS

The spooled files from the OS/400 to MVS were transferred over NJE via the VM/MVS Bridge. To initiate the sending different methods were used.

7.2.1 Print Request Functions

This section describes how to initiate a print request.

7.2.1.1 Using SNDNETSPLF for Print Request Submission

This is the normal way to initiate the sending of a spool file to an MVS system. This command can be entered on the command line of the OS/400 system, it can be initiated from WRKOUTQ panels, or it can be hidden in an application program. In each case, the final stage is the same; the SNDNETSPLF command with appropriate parameters is issued. There are different ways to create the spool data set. Normally, OS/400 applications produce SCS data stream, although it is also possible to create AFPDS print data sets.

There is a program called the Advanced Function Printing Utility to create some types of AFPDS in an OS/400 system. This program can print overlays and page segments on an AFP printer in OS/400 by creating a document calling these resources and including the resources as inline resources in front of the document. It can even print an OS/400 data base file using Printout Format Definitions. In each case, the resulting file is an AFPDS print data set.

The print data sets do not include all the parameters needed to specify the output characteristics. Neither is it possible to enter all these parameters in the SNDNETSPLF command.

It means that in most of the cases the print file will be printed in MVS using the default values specified on MVS side.

It is possible to transfer the print class and the form name to the MVS system. In MVS it is possible to write an user exit (APSUX07) to change the default values based on the print class and the form name information. There is an example of changing the default values for different form names. The example code is included in SAMPLIB with PSF/MVS. For more information about how to code the exit, see *PSF/MVS System Programmer's Guide*. Without user modification (coding exits APSUX04 and APSUX07) or applying rather recent PTFs to PSF/MVS, PSF/MVS does not accept AFP resources other than page definition and form definition inline in the print data set.

SCS data

SCS print data sets were sent to MVS by using SNDNETSPLF. In this case, data format *RCDDATA was used. The printing was done in the MVS system using the default values. The files were printed as expected.

Line data and structured field records

As there were no tools available to create a line data file with structured fields records imbedded, test files were created in the S/370 system and transferred to OS/400 to be printed by using PRTAFPDTA. The problem in this case is also the lack of tools to pass the AFP related information over to the MVS system. Thus, the names of the copy group and page format names as well as the page overlay names in the structured field records were made to match the names in the form definition and page definition in the MVS system to allow IMM, IDM and IPO records to be used. There is no similar problem with the page segments and IPS commands, as using IPS does not require the names of the segments to be specified in a form or page definition. The printed output was printed as expected in the MVS system.

Line data with inline PAGEDEF/FORMDEF

As there were no tools available to create a line data file with inline resources in OS/400, the test files were created in S/370 system and transferred to OS/400 to be printed by using PRTAFPDTA. The spool file created by the PRTAFPDTA command was then sent to the MVS system. The problem in this case is also the lack of tools to pass the AFP related information over to the MVS system. Thus, the names of the inline form definition and page definition were made to match the default values in MVS. The printed output was printed as expected in the MVS system.

Line data with inline fonts

As there were no tools available to create a line data file with inline resources in OS/400, the test files were created in S/370 system and transferred to OS/400 to be printed by using PRTAFPDTA. The spool file created by the PRTAFPDTA command was then sent to the MVS system. AFP resources other than form definition and page definition are not supported as inline resources by MVS. To allow other resources to be inline, two user exits were coded on the MVS side. With a rather recent PTF, this support is brought officially into PSF/MVS. With these modifications, the printed output was printed as expected in the MVS system.

Full AFPDS

Print files that are normal AFPDS documents print correctly except for the restrictions imposed by OS/400 capabilities to transfer AFP related parameters. The document was printed using the default form definition in the MVS system.

The data printed on the banner pages in the MVS system is missing all the necessary information to show which job originally sent the print file. The JOBNUMBER and JOBID fields constantly contain JOB00001 and AS400001. The only information referring to anything on the OS/400 side is NAME field that has the originating user ID printed.

As the information passed from the sending node is poor, it is not very useful, for example, for accounting purposes.

The user sending the print file to MVS gets a message in the message queue. This message only tells that the file has been received in the MVS node, but no message is sent when the printing has finished.

7.2.2 Migrating Resources from OS/400 to MVS

In many instances you will want to migrate print resources from the OS/400 system to the MVS platform. The following shows some of the ways you can accomplish this task.

As the AFP resources in OS/400 are in a format that is not accessible by normal programming means, the migrating of the resources is rather difficult.

It may be possible to use Machine Interface and C/400 with PRPQ P10102 to access the resources. It was not possible to test this during the residency.

So, even migrating the resources using a magnetic tape as the media needs much programming work.

It is not possible to send the resources as normal files to other systems, where more tools might be available. So, even this alternative requires programming.

If SAA PrintManager is available, it would be possible to use its inline resource functions to send a file with inline resources to the MVS system. Then it would be possible to extract the resources from the spool file with rather moderate programming effort.

The OS/400 command DMPOBJ can access these resources, so it can be used to find a laborious way to migrate the resources. The procedure starts with dumping the resource with DMPOBJ. The output will be in the output queue QPSRVDMP. The dump has all the data included in the resource printed in hexadecimal. This spool file can be sent to another node using the SNDNETSPLF command. In the receiving node this print file is received to a file. With a REXX EXEC or a program written in some other language, this spool file can be changed to a resource. Obviously this works, but the procedure is far from an easy and user-friendly way to do it. We had no time to write the EXEC or program code to accomplish the test.

By using AFPU to send page segments or overlays inline, resources can be migrated. The file created by AFPU can be sent to a user ID in MVS, received to a file in MVS, and then with an EXEC (see A.2, “Routine to Extract AFP Inline Resources” on page 76) create a resource from the file. The AFPU program does not support resources other than form definitions, page segments and

overlays. It is not very likely that unique fonts exist in AS/400 system, so the need to send these objects is rather rare.

In the most recent releases, there is a function to transform an AFP resource to a physical file member. This transformed file can then be sent through the network, or it can be moved manually using a magnetic tape to the MVS system and stored into an AFP resource library there.

7.3 Printing from OS/400 to VM

This section describes printing from OS/400 on a VM system. The spooled files from OS/400 to VM were transferred over NJE via the VM/MVS Bridge.

7.3.1 Print Request Functions

This section describes how to initiate a print request.

7.3.1.1 Using SNDNETSPLF for Print Request Submission

This is the normal way to initiate the sending of a spool file to a VM system. This command can be entered on the command line of the OS/400 system, it can be initiated from WRKOUTQ panels, or it can be hidden in an application program. In each case, the final stage is the same; the SNDNETSPLF command with appropriate parameters is issued. There are different ways to create the spool data set. Normally, OS/400 applications produce SCS data stream, although it is also possible to create AFPDS print data sets.

There is a program called the Advanced Function Printing Utility to create some types of AFPDS in an OS/400 system. This program can print overlays and page segments on an AFP printer in OS/400 by creating a document calling these resources and including the resources as inline resources in front of the document. It can even print an OS/400 data base file using Printout Format Definitions. In each case, the resulting file is an AFPDS print data set. This approach works for VM, as VM accepts all resources as inline resources.

The print data sets do not include all the parameters needed to specify the output characteristics. Neither is it possible to enter all these parameters in SNDNETSPLF command.

It means that in most of the cases the print file will be printed in VM using the default values specified on the VM side.

It is possible to transfer the print class and the form name to the VM system. Based on form name or print class, it is possible in PSF/VM to set different default values for all AFP related parameters. This enables more AFP related information passed from OS/400 to VM. To have the expected result, an agreement on print classes and form names and the parameters related to these has to exist between OS/400 and VM systems.

SCS data

SCS print data sets were sent to VM by using SNDNETSPLF. In this case, data format *RCDDATA was used. The printing was done in the VM system using the default values. The files were printed as expected.

Line data and structured field records

As there were no tools available to create a line data file with structured fields records imbedded, test files were created in the S/370 system and transferred to OS/400 to be printed by using PRTAFPDTA. The problem in this case is also the lack of tools to pass the AFP related information over to the VM system. Thus, the names of the copy group and page format names as well as the page overlay names in the structured field records were made to match the names in the form definition and page definition in the VM system to allow IMM, IDM and IPO records to be used. There is no similar problem with the page segments and IPS commands, as using IPS does not require that the names of the segments be specified in a form or page definition. The printed output was printed as expected in the VM system.

Line data with inline PAGEDEF/FORMDEF

As there were no tools available to create a line data file with inline resources in OS/400, the test files were created in the S/370 system and transferred to OS/400 to be printed by using PRTAFPDTA. The spool file created by the PRTAFPDTA command was then sent to the VM system. The problem in this case is also to lack of tools to pass the AFP related information over to the VM system. Thus the names of the inline form definition and page definition were made to match the default values in VM. The printed output was printed as expected in the VM system.

Line data with inline fonts

As there were no tools available to create a line data file with inline resources in OS/400, the test files were created in the S/370 system and transferred to OS/400 to be printed by using PRTAFPDTA. The spool file created by the PRTAFPDTA command was then sent to the VM system. The printed output was printed as expected in the VM system.

Full AFPDS

Print files that are normal AFPDS documents print correctly except for the restrictions imposed by OS/400 capabilities to transfer AFP related parameters. The document was printed using the default form definition in the VM system.

The data printed on the banner pages in the VM system is missing most of the necessary information to show which job originally sent the print file. The USERID and NODEID show correctly the user id and node id, where the print file was sent from. It also shows in FILETYPE field the name of the original spool file in OS/400.

As the information passed from the sending node is poor, it is not very useful, for example, for accounting purposes.

The user sending the print file to VM gets a message in the message queue. This message indicates that the file has been received in the VM node. PSF/VM SFCM and PDM message files are returned to the user in OS/400 after PSF/VM has finished the printing of the file.

7.3.2 Migrating Resources from OS/400 to VM

In many instances, you will want to migrate print resources from the OS/400 system to the VM platform.

As the AFP resources in OS/400 are in a format that is not accessible by normal programming means, the migrating of the resources is rather difficult.

It may be possible to use Machine Interface and C/400 with PRPQ P10102 to access the resources. It was not possible to test this during the residency.

So, even migrating the resources using a magnetic tape as the media needs much programming work.

It is not possible to send the resources as normal files to other systems, where more tools might be available. So, even this alternative requires programming.

If SAA PrintManager is available, it would be possible to use its inline resource functions to send a file with inline resources to the VM system. As VM accepts any resources inline, there may be no need to put them in the libraries of the VM system. Anyway, it would be possible to extract the resources from the spool file with rather moderate programming effort.

The OS/400 command DMPOBJ can access these resources, so it can be used to find a laborious way to migrate the resources. The procedure starts with dumping the resource with DMPOBJ. The output will in the output queue QPSRVDMP. The dump has all the data included in the resource printed in hexadecimal. This spool file can be sent to another node using the SNDNETSPLF command. In the receiving node, this print file is received to a file. With a REXX EXEC or a program written in some other language, this spool file can be changed to a resource. Obviously this works, but the procedure is far from an easy and user-friendly way to do it. We had no time to write the EXEC or program code to accomplish the test.

By using AFPU to send page segments or overlays inline, resources can be migrated. The file created by AFPU can be sent to a user ID in VM, received to a file in VM, and then with an EXEC (see B.2, “OS/400 Resource Converter for VM” on page 158) create a resource from the file. The AFPU program does not support resources other than form definitions, page segments and overlays. It is not very likely that unique fonts exist in AS/400 system, so the need to send these objects is rather rare.

In the most recent releases, there is a function to transform an AFP resource to a physical file member. This transformed file can then be sent through the network, or it can be moved manually using a magnetic tape to the VM system and stored onto a CMS disk there.

7.4 Printing from OS/400 to VSE

This section describes printing from OS/400 on a VSE system. The spooled files from OS/400 to VM were transferred over NJE via the VM/MVS Brigde. The spooled files from the OS/400 to the VSE were transferred first through the VM/MVS Bridge to a VM node and then with RSCS to VSE.

A direct connection between OS/400 and VSE has been successfully tested, but this is not officially supported.

7.4.1 Print Request Functions

This section describes how to initiate a print request.

7.4.1.1 Using SNDNETSPLF for Print Request Submission

This is the normal way to initiate the sending of a spool file to an VSE system. This command can be entered on the command line of the OS/400 system, it can be initiated from WRKOUTQ panels, or it can be hidden in an application program. In each case the final stage is the same; the SNDNETSPF command with appropriate parameters is issued. There are different ways to create the spool data set. Normally, OS/400 applications produce SCS data stream, although it is also possible to create AFPDS print data sets.

There is a program called the Advanced Function Printing Utility to create some types of AFPDS in an OS/400 system. This program can print overlays and page segments on an AFP printer in OS/400 by creating a document calling these resources and including the resources as inline resources in front of the document. It can even print an OS/400 data base file using Printout Format Definitions. In each case, the resulting file is an AFPDS print data set. This approach does not work for VSE except for files that have no inline resources or have only VSE supported inline resources included.

The print data sets do not include all the parameters needed to specify the output characteristics. Neither is it possible to enter all these parameters in SNDNETSPLF command.

It means that in most of the cases the print file will be printed in VSE using the default values specified on the VSE side.

It is possible to transfer the form name to the VSE system. In VSE, it is possible to include AFP related parameters in a printer parameter member with the name of a form (prefixed by Z1). This would make it possible to define different sets of defaults to be used for print jobs from an OS/400 system. PSF/VSE does not accept AFP resources than page definition and form definition inline in the print data set.

SCS data

SCS print data sets were sent to VSE by using SNDNETSPLF. In this case, data format *RCDDATA was used. The printing was done in the VSE system using the default values. The files were printed as expected.

Line data and structured field records

As there were no tools available to create a line data file with structured fields records imbedded, test files were created in the S/370 system and transferred to AS/400 to be printed by using PRTAFPDTA. The problem in this case is also the lack of tools to pass the AFP related information over to the VSE system. Thus, the names of the copy group and page format names as well as the page overlay names in the structured field records were made to match the names in the form definition and page definition in the VSe system to allow IMM, IDM and IPO records to be used. There is no similar problem with the page segments and IPS commands, as using IPS does not require that the names of the segments be specified in a form or page definition. The printed output was printed as expected in the VSE system.

Line data with inline PAGEDEF/FORMDEF

As there were no tools available to create a line data file with inline resources in AS/400, the test files were created in the S/370 system and transferred to AS/400 to be printed by using PRTAFPDTA. The spool file created by the PRTAFPDTA command was then sent to the VSE system. The problem in this case is also the lack of tools to pass the AFP related information over to the VSE system. Thus, the names of the inline form definition and page definition were made to match the default values in VSE. The printed output was printed as expected in the VSE system.

Line data with inline fonts

As VSE does not accept fonts as inline resources, this case was not tested.

Full AFPDS

Print files that are normal AFPDS documents print correctly except for the restrictions imposed by OS/400 capabilities to transfer AFP related parameters. The document was printed using the default form definition in the VSE system.

The data printed on the banner pages in the VSE system is missing all the necessary information to show which job originally sent the print file. The JOB NO field always contains 00001, and the JOBNAME is AS400001. The only information referring to anything on the OS/400 side, is the ORG USER field that has the originating user ID printed and NODE that indicates the node where the file was sent from.

As the information passed from the sending node is poor, it is not very useful, for example, fot accounting purposes.

The user sending the print file to VSE gets a message in the message queue. This message only indicates that the file has been received in the VSE node, but no message is sent when the printing has finished.

7.4.2 Migrating Resources from OS/400 to VSE

In many instances, you will want to migrate print resources from the OS/400 system to the VSE platform. The following shows some of the ways you can accomplish this task.

As the AFP resources in OS/400 are in a format that is not accessible by normal programming means, the migrating of the resources is rather difficult.

It may be possible to use Machine Interface and C/400 with PRPQ P10102 to access the resources. It was not possible to test this during the residency.

So, even migrating the resources using a magnetic tape as the media needs much programming work.

It is not possible to send the resources as normal files to other systems, where more tools might be available. So, even this alternative requires programming.

If SAA PrintManager is available, it would be possible to use its inline resource functions to send a file with inline resources to the VSE system. Then it would be possible to extract the resources from the spool file with rather laborious programming effort.

The OS/400 command DMPOBJ can access these resources, so it can be used to find a laborious way to migrate the resources. The procedure starts with dumping the resource with DMPOBJ. The output will in the output queue QPSRVDMP. The dump has all the data included in the resource printed in hexadecimal. This spool file can be sent to another node using the SNDNETSPLF command. In the receiving node, this print file is placed to the POWER queue. With a user written program, this spool file can be read from the POWER list queue and changed to a resource in the VSE library. Obviously this works, but the procedure is far from an easy and user-friendly way to do it. We had no time to write the program code to accomplish the test.

By using AFPU to send page segments or overlays inline, resources can be migrated. The file created by AFPU can be sent to the VSE system. A program can be written to extract the resource from the spool file in the POWER/VSE queue, but this is a very complicated way to do it. The AFPU program does not support resources other than form definitions, page segments and overlays. It is not very likely that unique fonts exist in AS/400 system, so the need to send these objects is rather rare.

In the most recent releases, there is a function to transform an AFP resource to a physical file member. This transformed file can then be moved manually using a magnetic tape to the VSE system and, by using a user program, create an AFP resource into the VSE library.

7.5 Printing from OS/400 to OS/400

In our test case two OS/400 systems were connected with an SDLC line.

7.5.1 Print Request Functions

This section describes how to initiate a print request.

7.5.1.1 Using SNDNETSPLF for Print Request Submission

This is the normal way to initiate the sending of a spool file to another OS/400 system. This command can be entered on the command line of the OS/400 system, it can be initiated from WRKOUTQ panels, or it can be hidden in an application program. In each case, the final stage is the same; the SNDNETSPLF command with appropriate parameters is issued. There are different ways to create the spool data set. Normally, OS/400 applications produce SCS data stream, although it is also possible to create AFPDS print data sets.

There is a program called the Advanced Function Printing Utility to create some types of AFPDS in an OS/400 system. This program can print overlays and page segments on an AFP printer in OS/400 by creating a document calling these resources and including the resources as inline resources in front of the document. It can even print an OS/400 data base file using Printout Format Definitions. In each case, the resulting file is an AFPDS print data set. This approach works when sending to another OS/400 system, as OS/400 AFP accepts any resources as inline resources.

The print data sets do not include all the parameters needed to specify the output characteristics. Neither is it possible to enter all these parameters in SNDNETSPLF command. Specifying the *ALLDATA parameter in the data format field causes all information relevant to that print file to be sent over to the other OS/400 system.

SCS data. Worked as expected.

Line data and structured field records. Worked as expected.

Line data with inline PAGEDEF/FORMDEF. Worked as expected.

Line data with inline fonts. Worked as expected.

Full AFPDS. Worked as expected.

All the test cases were printed correctly. It is not possible, without major programming work, to create all the test cases in OS/400. Because of that, files that were created and then transferred to OS/400, were mostly used as test cases.

In some cases, output from the AFP Utility was used as a test print file.

The data printed on the banner pages in the printing OS/400 contain, only information from the printing system and does not identify the real origin of the print file.

As the information passed from the sending node is poor, it is not very useful, for example, for accounting purposes.

TCP/IP implementation in OS/400 has functions that allows the user to send almost any print file to another OS/400 system. This was included in the new version of the OS/400 operating system.

7.5.2 Migrating Resources from OS/400 to OS/400

In many instances, you will want to migrate print resources from the OS/400 system to the OS/400 platform. The following shows some the way you can accomplish this task.

The example is from the *IBM AS/400 Printing II*.

- Create a Save File.
- Save the resource to the save file.
- Send the save to the other OS/400.
- RCVNETF on the receiving system.
- Restore the objects to a data base.

7.6 Printing from OS/400 to OS/2

This section describes printing of OS/400 files on a printer driven by PSF/2 on an OS/2 server.

In general, the facilities that support communications between OS/400 and OS2 do not facilitate printing. There is, for instance, no direct spool-to-spool communication of print files, such as Network Job Entry (NJE) provides for MVS, VM, VSE, and OS/400.

The Distributed Print Function included in PSF/2 Release 1.10 enables an OS/2 attached PSF/2 supported printer to act as a printer to the OS/400 system. PSF/2 receives the print file to an IPDS spool file in OS/2 and prints it on a printer in

the order set by the system administrator. In this case, this is not really cross-system printing, although the system controlling the printing is different from the system driving the physical device.

AS/400 TCP/IP implementation provides some functions that allow the user to send an OS/400 spool file to an OS/2 system to be printed there.

The first requirement is a vehicle to move the print file from OS/400 to OS2. There are not many possibilities:

- OS/2 Communications Manager 3270 Emulation File Transfer Program

  An OS/2 user that is logged on to the OS/400 using OS/400 PC Support can issue an OS/2 command that will transfer a file of data from OS/400 to the OS/2 workstation.

- TCP/IP File Transfer Protocol

  An OS/400 user can use the TCP/IP FTP program to “login” to the OS/2 server and ship a file of data.

The second requirement is a mechanism to deliver the print data to the spool of the target OS/2 system, accompanied by the control information necessary to govern the printing process. Both processes described above require user interaction (that is, an APRINT command) on the OS/2 machine in order to complete the printing task.

7.6.1 Technical Hurdles

There are some technical hurdles associated with printing OS/400 files on an OS/2 server:

- OS/2 represents data in ASCII while OS/400 uses EBCDIC code points. This means that normal text data must go through an EBCDIC to ASCII translation during transport. Text files that contain unusual code points may not be accurately translated. For instance, line-data files that contain machine carriage control characters may not be properly translated by the standard translate tables.

- AFPDS data, on the other hand, is exactly the same on both OS/400 and OS/2. That is, when transporting AFPDS data from OS/400 to OS/2, no conversion from EBCDIC to ASCII should be done (that is, BINARY transfer). Therefore, it is not possible to transport files that contain both AFPDS and non-AFPDS data. The translation option chosen applies to the entire file which means it will be incorrect for part of the data.

- OS/2 has no page definition concept. The page definition is used by PSF to map line-data files to AFP pages. This mapping depends upon 1403/3800-1 line-data structures which are not native to OS/2. Therefore, PSF/2 does not employ page definitions, nor does it support OS/400 SCS print files, just as PSF on the OS/400 does not support the printing of ASCII print files.

  The lack of page definition support on OS/2 has some deeper implications:

  - Many of the simple page formats OS/400 users expect to be able to apply to print files, such as 2-up or landscape printing, are not available when the file is sent to OS/2.

  - It is not possible to select a character set (font) using the CHARS parameter.

  - Carriage control and TRC characters are not recognized by OS/2.

– Conditional processing is not available.

There is no LN2AFPDS program available in an OS/400 environment, so there is no way around the problems above.

7.6.2 Print Request Functions

This section describes how to receive a print data file from an OS/400 system to an OS/2 system and print it in the OS/2 system.

7.6.2.1 Using File Transfer to Download the Print File

If the file to be transmitted and printed is a flat file without carriage control characters and TRC characters, the file can be transmitted using EBCDIC/ASCII conversion in the file transfer. After the file has been received, it is possible to print it using the APRINT command with the appropriate parameters.

If the file is an AFPDS file, it is downloaded with the BINARY option and then printed using the APRINT command, again setting up the parameters as needed.

If the file is a mixture of line data and AFP data, or if the file is a line data file with either carriage control or TRC bytes, the file has to be converted to AFPDS to achieve a correct output. There are no tools available for this in an OS/400 system. This conversion needs a major programming effort by the user. After conversion, the file would be in AFPDS format, and can be received to the OS/2 system and then printed in the same way as any AFPDS file.

7.6.3 Print Resource Migration from OS/400 to OS/2

There may be cases where the receiving PSF/2 does not have all the resources required to print the job. In such cases, the resources must either be placed inline with the print file, or they must be placed in the PSF/2 resource libraries before the print request is submitted. Since AFP resources are AFPDS structures, under OS/2 they may only be used with full AFPDS files, for the reasons discussed as “technical hurdles” on page 59. Flat files that require some special resource can only be handled by preinstalling that resource in the PSF/2 resource library.

AFPDS resources may be transferred from OS/400 to OS/2 using 3270 File Transfer if the resources can be converted to physical file members. In the most recent releases, there is a function to transform an AFP resource in an OS/400 system to a physical file member. This transformed file can be downloaded to the OS/2 system. The transfer must be ***BINARY*** to prevent the destruction of AFPDS data. As AS/400 pads all the records with blanks, the resource has to be changed before it is usable in the PSF/2 system. A program to remove the extra blanks is in D.1, “AS4002OS Routine to Remove Extra Blanks” on page 189. The resulting OS/2 file must then be added into the PSF/2 resource library using an RLADD command.

Chapter 8. Printing from an OS/2 Host

This chapter describes some possible implementations for AFP printing from an OS/2 system. It discusses the different mechanisms for requesting and transmitting a print request to a target system.

In general, no available facilities capable of connecting an OS/2 workstation to something else provide a solid foundation for implementing automated handling services for AFP print work. There is, for instance, no direct spool-to-spool communication of print files, such as Network Job Entry (NJE) provides for MVS, VM, VSE, and AS/400. Some specific product environments, such as OfficeVision* and Enhanced Connectivity Facilities (ECF) offer support for their own requirements, but there is no generic AFP print file support that is available to a wide array of products and applications.

Some of the connections available to an OS/2 workstation:

- LAN Services

  Local Area Networks are a very common method of joining OS/2, DOS, AIX, and other types of workstation together. LANs provide the capability for users to send messages to one another and share data and services.

  Users on host mainframes, however, have no natural access to LAN-based services. While it is true that host mainframes may be attached to LANs and use them as communication vehicles, end users on the mainframe usually have access only to some particular LAN-based application that has been specifically coded to communicate with their host application.

- APPC

  Advanced Program-to-Program Communication offers a set of facilities that provide application program access to the SNA LU 6.2 protocol for program to program communication. While it provides a well defined vehicle for communication between a host mainframe application and an OS/2 application, it requires that those applications be written. APPC provides programming interfaces only.

- Host Terminal Emulators

  There are a number of 3270 terminal emulation packages available, including the emulation services provided by the OS/2 Communication Manager. These emulators offer the capability to move data files between the host mainframe and the OS/2 workstation, in both directions.

  These emulator packages are designed, however, to permit an end user at the workstation to use it as a host terminal. Therefore, the facilities are designed to require human interaction. The workstation must be logged on to a host interactive session before files may be transferred. Further, all file transfer activity must be initiated from the workstation.

- TCP/IP

  TCP/IP provides protocols for connecting peer systems together that have become a de facto industry standard over the past decade. It offers the benefit of being designed to connect disparate systems together in a transparent way. In addition to all IBM SAA platforms being supported, TCP/IP permits connections to AIX and UNIX systems. Further, most implementations provide some higher level application functions, such as

© Copyright IBM Corp. 1994

File Transfer Protocol, which moves files between systems, TELNET, which provides remote logon service, and Simple Mail Transfer Protocol which provides electronic mail services.

FTP provides facilities for files to be moved from one TCP/IP node to another. Since both systems are peers, transport may be initiated by either end. This fact makes it highly attractive as a vehicle for moving print files from any client to any server.

There is also a common TCP/IP application known as Remote Print, usually accessed using the command LPR. This command provides a neat user interface for shipping print files to a printer driven by a remote system.

During the course of our project, we evaluated the feasibility of basing some generalized server function on these available vehicles:

- LAN Server/Requester facilities are, of course, unavailable to host users.
- APPC offers the necessary facilities, but requires extensive user programming to access them.
- The 3270 Emulation File Transfer Program cannot support a generalized server function because file transfer can only be initiated from one end.
- The TCP/IP option carries the powerful attraction that it permits connection between a wide variety of operating platforms, both IBM and non-IBM. The File Transfer Protocol function requires some user programming to make it useful as a carrier of AFP print work, but far less than would be the case with APPC. In addition, TCP/IP will connect to more platforms than APPC.

8.1.1 File Transfer Protocol Technical Hurdles

In implementing our TCP/IP File Transfer Protocol server, we encountered some technical obstacles. Those germane to the OS/2 platform are:

- OS/2 represents data in ASCII while MVS, VM, VSE, and OS/400 use EBCDIC code points. This means that normal text data must go through an ASCII to EBCDIC translation during transport to one of these systems. File Transfer Protocol provides for either BINARY transport, which does no code point translation, or ASCII transport, which translates ASCII to EBCDIC during transmission.

  Text files that contain unusual code points may not be accurately translated. For instance, line-data files that contain machine carriage control characters may not be properly translated by the standard FTP translate tables. One would not normally expect an OS/2 application to generate host line-data output, but it is quite possible that such a file might arrive at the OS/2 system from elsewhere.

- AFPDS data, on the other hand, is exactly the same on all platforms. That is, when transporting AFPDS data from OS/2 to an EBCDIC platform, no conversion from ASCII to EBCDIC should be done (that is, BINARY transfer). Therefore, it is not possible to transport files that contain both AFPDS and non-AFPDS data. The translation option chosen applies to the entire file which means it will be incorrect for part of the data.

- OS/2 has no concept of a ***record***, whereas MVS, VM, VSE, and OS/400 expect data files to comprise records. Files under OS/2 are a continuous stream of characters. When such files are transported to one of these systems, they must be reorganized into records. For text files this is done by breaking the data stream into records based on the presence of Carriage Return/Line

Feed (CRLF) sequences. Because AFPDS must be transported to the host using BINARY transmission (that is, no translation from ASCII to EBCDIC), the records cannot be identified by the presence of CRLF sequences. Therefore, AFPDS is broken into records by filling each record to the maximum defined record length, without regard to the lengths of the actual AFPDS structured fields. In the final result, some records may contain many AFPDS structured fields, and, conversely, some AFPDS structured fields may span multiple records. Such a file is not printable by PSF, which expects each AFPDS structured field to occupy a single variable length record. The MVS AFPDSFIX routine, described in A.1, “AFPDSFIX Routine” on page 73, provides facilities to reconstruct AFPDS records that have been “streamed” in this fashion.

8.2 Printing from OS/2 to MVS

This section describes printing of files from an OS/2 workstation on a printer driven by PSF/MVS on an MVS system.

8.2.1 Print Request Functions

This section describes how to upload a print file from an OS/2 system to an MVS system.

There is no automatic way to upload a print file from an OS/2 system to an MVS system.

The user has to upload the file manually using terminal emulator session or TCP/IP FTP.

After uploading, the file can be printed in the MVS system with the utilities available in the MVS system.

A flat file from an OS/2 system can be uploaded using ASCII/EBCDIC conversion provided that the file does not contain any printer control characters.

An AFPDS file can be uploaded, and it has to be uploaded as binary. The differences in the file structure in OS/2 and MVS causes one extra step. An AFPDS file in OS/2 is a file without a record structure, as if it were one long record. In MVS it has to be split into records before PSF/MVS accepts it. One way to do this splitting is described in A.1, “AFPDSFIX Routine” on page 73.

8.2.2 Print Resource Migration from OS/2 to MVS

There may be cases where the receiving MVS system does not have all the resources required to print the job. In such cases, the resources must either be placed inline with the print file, or they must be placed in the MVS system′s resource libraries before the print request is submitted. Since AFP resources are AFPDS structures, under OS/2 they may only be used with full AFPDS files, for the reasons discussed as 8.1.1, “File Transfer Protocol Technical Hurdles” on page 62. Flat files that require some special resource can only be handled by preinstalling that resource in the MVS libraries.

AFPDS resources may be transferred from OS/2 to MVS using either 3270 File Transfer, or the TCP/IP File Transfer Protocol. In both cases, the transfer must be ***BINARY*** to prevent the destruction of AFPDS data. In both cases, the resulting file will *not* be acceptable to PSF/MVS because of the “streaming” of

the AFPDS discussed as 8.1.1, “File Transfer Protocol Technical Hurdles” on page 62. The AFPDSFIX routine, documented in A.1, “AFPDSFIX Routine” on page 73 must be run against the resource on the MVS system before it is moved into the PSF/MVS resource libraries.

The manual *PSF/2 Type Transformer User's Guide* has some information about how to migrate resources from an OS/2 system to the MVS system by using the tools included in PSF/2 Type Transformer.

8.3 Printing from OS/2 to VM

This section describes printing of files from an OS/2 workstation on a printer driven by PSF/VM on a VM system.

8.3.1 Print Request Functions

This section describes how to upload a print file from an OS/2 system to a VM system.

There is no automatic way to upload a print file from an OS/2 system to a VM system.

The user has to upload the file manually using a terminal emulator session or TCP/IP FTP.

After uploading, the file can be printed in the VM system with the commands available in the VM system.

A flat file from an OS/2 system can be uploaded using ASCII/EBCDIC conversion provided that the file does not contain any printer control characters.

An AFPDS file can be uploaded, and it has to be uploaded as binary. The differences in the file structure in OS/2 and VMS causes one more problem. An AFPDS file in OS/2 is a file without a record structure, as if it were one long record. In VM it has to be split into records before PSF/VM accepts it. One way to do this splitting is described in B.1, “AFPDSFIX routine for VM” on page 155.

8.3.2 Print Resource Migration from OS/2 to VM

There may be cases where the receiving VM system does not have all the resources required to print the job. In such cases, the resources must either be placed inline with the print file, or they must be placed in the VM system′s resource libraries before the print request is submitted. Since AFP resources are AFPDS structures, under OS/2 they may only be used with full AFPDS files, for the reasons discussed as 8.1.1, “File Transfer Protocol Technical Hurdles” on page 62. Flat files that require some special resource can only be handled by preinstalling that resource in the VM libraries.

AFPDS resources may be transferred from OS/2 to VM using either 3270 File Transfer, or the TCP/IP File Transfer Protocol. In both cases, the transfer must be ***BINARY*** to prevent the destruction of AFPDS data. In both cases, the resulting file will NOT be acceptable to PSF/VM because of the “streaming” of the AFPDS discussed as 8.1.1, “File Transfer Protocol Technical Hurdles” on page 62. The AFPDS_FIX subroutine coding documented in B.1, “AFPDSFIX routine for VM” on page 155 is used to reconstruct AFPDS print files that have

been “streamed” by OS/2. The AFPDS resources shipped to VM from OS/2 must be run through a similar routine to make them usable by PSF/VM.

The manual *PSF/2 Type Transformer User's Guide* has some information about how to migrate resources from an OS/2 system to the VM system by using the tools included in PSF/2 Type Transformer.

8.4 Printing from OS/2 to VSE

This section describes printing from OS/2 on a VSE system.

No IBM product available at the time of this writing provides support for the automatic printing of OS/2 files on a printer driven by PSF/VSE.

In Power IWS (Intelligent Workstation Services) there are some restricted ways to print from OS/2 or DOS directly to a printer attached to VSE. These methods require that the user have a session to CICS in VSE. Using programs that have been downloaded from the VSE system, the user can send a file into the POWER spool. The file can be directly sent to the POWER list (LST) queue, or the file can be sent to POWER reader (RDR) queue to be executed as a batch job in VSE. With these methods, almost any print file originating from the workstation can be sent to VSE and printed on a printer attached to VSE.

As VSE systems are rather often running under VM, it might be possible to use the VM system as an intermediate node.

Then all the methods used in OS/2 to VM printing and then VM to VSE printing, are possible.

8.4.1.1 Using POWER IWS for Print Request Submission

With VSE, a set of programs to enable moving files from an intelligent workstation to the VSE POWER queues and from the POWER queues to the workstation is provided.

The file is sent from an OS/2 system to the POWER list queue by issuing, for example, the following command:

```
SEND PCFILE HOSTLIST (FILE=LST
```

This would send a PC file called PCFILE to the POWER list queue and to have the name HOSTLIST in the POWER list queue.

This method works for a flat file with no control characters. It does not work for other files. Sending, for example, AFP files from an OS/2 workstation to a VSE system needs a lot of programming.

8.4.2 Print Resource Migration

To migrate resources from an OS/2 system to a VSE system needs a lot of programming. An example of how to do this is described in Appendix C, “VSE AFP Sample Programs” on page 163.

The manual *PSF/2 Type Transformer User's Guide* has some information about how to migrate resources from an OS/2 system to the VSE system by using the tools included in PSF/2 Type Transformer.

8.5 Printing from OS/2 to OS/400

This section describes printing from OS/2 to an OS/400 system.

OS/400 PC support provides tools to initiate a print job from the OS/2 system to the OS/400 system. From the user point of view a printer attached to the OS/400 system is as any printer locally attached to the OS/2 system. OS/400 PC support accepts all kinds of data streams including PC ASCII and AFPDS.

The functions of OS/400 PC support are described in detail in the document *IBM AS/400 Printing III*.

8.5.1 Migration of Print Resources

OS/400 PC support allows the user to send OS/2 resources to an OS/400 system.

The CRT commands in OS/400 expect that the resource is in a physical file member. The format of the file has to be fixed length records and the records must be padded with blanks. A program (see D.2, “Program to Pad a Resource with Blanks” on page 190) was written to transform an AFP resource in the OS/2 library to a format with fixed length records padded with blanks.

After this transformation, by using OS/400 PC Support, the resource can be uploaded to a physical file member in OS/400. After uploading, the resource can be created with an appropriate CRT command.

The manual *PSF/2 Type Transformer User's Guide* has some information about how to migrate resources from an OS/2 system to the OS/400 system by using the tools included in PSF/2 Type Transformer.

8.6 Printing from OS/2 to OS/2

This section describes the printing of files from an OS/2 workstation on a printer driven by the PSF/2 print driver.

Local Area Networks are by far the most prevalent method of connecting OS/2 systems together. Therefore, we will discuss only the submission of print work from a LAN requester to a print server machine. While it is true that users might elect to run PSF/2 on their own workstations, it is unlikely that this will be common. The RAM, disk, and processor resources required to operate PSF/2 do not make it the driver of choice for a personal printer. PSF/2 is designed to drive multiple shared printers in a server machine.

PSF/2 provides two mechanisms for submitting print work:

1. The APRINT command that may be entered from any OS/2 command prompt.
2. A Print Submitter PM application that provides a window interface for the user. Requests built with this facility actually result in an APRINT command. Therefore, we will deal only with the APRINT command in the discussion below.

The user interface provided by PSF/2 permits the user to specify AFP parameters, such as DUPLEX, BIN, COPIES, and FORMDEF.

In addition, to the submission facilities supplied by PSF/2, there are a number of other ways to direct work to a PSF/2 printer:

- DOS PRINT to the device associated with the PSF/2 queue.
- OS2 COPY > to the device associated with the PSF/2 queue.
- OS2 TYPE > to the device associated with the PSF/2 queue.
- Assign the PSF/2 queue as the application default. This will cause all undirected Presentation Manager print requests to be assigned to the PSF/2 queue. Printing a HELP screen would be an example of this.

All of the above amount to directing or redirecting the file to an OS2 queue or device served by PSF/2.

8.6.1 Print Request Functions

This section describes two ways to initiate a print request.

8.6.1.1 Using APRINT for Print Request Submission

APRINT is an OS/2 command that is provided with the PSF/2 product. It may be issued from any OS/2 command prompt using the following general syntax:

```
APRINT filename DEST=queuename  PARM1= PARM2= ...
```

For example, the following:

```
aprint c:\config.sys dest=pr3820 copies=3 duplex=yes
```

would result in three duplexed copies of the CONFIG.SYS file being printed on the printer serving the PrintManager Queue named PR3820.

PSF/2 also includes a Presentation Manager application that provides an interactive interface for the APRINT command.

In order for these facilities to be accessible to workstations other than the one running PSF/2, the PSF/2 server must be properly defined as a LAN server machine, and the PSF/2 print queues and directories must be defined as shared resources.

Additional Considerations

- **User Notification**

  If the MESSENGER and NETPOPUP services are running, a message popup is sent to the requester when the printing is finished. Unfortunately, the message always indicates that the file has been deleted and did not print. This effect is caused by the fact that PSF/2 first selects the file from the PrintManager queue, puts it back in hold status, and then deletes it when the printing has been successfully completed. PSF/2 must do this because the OS/2 spooler lacks the facilities to permit proper recovery of the printing in the event of an error. Therefore, PSF/2 must retain a copy of the print file in the queue by using the hold technique. An unfortunate side effect of this is that when PSF/2 finally deletes the job when the printing is complete, the OS2 PrintManager sends a message to the requester that the file has been deleted and did not print.

- **Banner Information**

  Several APRINT parameters affect the information which is displayed on the banner page:

  – **Jobowner:** This parameter permits the specification of both a user ID and a node ID. These are displayed in several places on the banner

page. The user ID will be displayed in the large block letters in the center of the main box.

  - **Jobname:** The text coded for this parameter will appear in sub-box under the main box of the banner page.
  - **Distribution:** The text coded for this parameter will appear in the upper right corner, next to user ID and node ID.

- **Accounting/Audit Information**

  PSF/2 provides an exit point for the extraction of auditing and accounting information. No standard code is provided for this exit. Customers may write their own routines to capture the information they deem useful.

Discussion of Data streams: Please see the definitions provided on page 1.3, "Test Cases" on page 2 for descriptions of the data stream types referenced in the following items.

- **Flat file**

  ASCII flat files generally print correctly with no special parameters needed. PSF/2 attempts to identify the type of input data file it receives and handle it appropriately. Since it is possible, albeit unlikely, that an ASCII file might accidentally appear to conform to valid AFPDS structures, try using the DATATYPE=ASCII parameter on the APRINT command if a file prints incorrectly.

- **Full AFPDS**

  Full AFPDS files print with full support of all AFP functions. No special APRINT parameters are required for AFPDS files. If the file contains inline resources, they will be used. All resource types, with the exception of page definition, are supported inline.

  Note: It is not possible to mix AFPDS and non-AFPDS data in the same print file. When PSF/2 encounters the first data that is not valid AFPDS structures, it terminates the print job.

- **Presentation Manager Metafile**

  OS/2 Presentation Manager applications may generate Presentation Manager Metafiles. Often they contain graphics. They print with no special considerations.

- **QuietWriter ASCII data**

  Any ASCII print file generated using the QuietWriter3 level of printing escape sequences, or a subset thereof [1], will print with no special considerations. You must be aware, however, that this support is an emulation of QuietWriter function and does not support everything in exactly the same fashion as a real QuietWriter.

  PSF/2 attempts to identify the type of input data file received and handle it appropriately. Since it is possible, albeit unlikely, that an ASCII file might accidentally appear to conform to valid AFPDS structures, try using the DATATYPE=ASCII parameter on the APRINT command if a file prints incorrectly.

[1] ProPrinter ASCII is a subset of QuietWriter ASCII.

8.6.1.2 Using File Redirection for Print Request Submission

This section discusses the use of some traditional OS/2 and DOS techniques to send print data to a PSF/2 server.

- **Using DOS PRINT**

  DOS users often use the PRINT command to direct print work to a specific print device. For example:

  ```
  print config.sys /d:lpt1
  ```

  could be used to print file CONFIG.SYS on the printer attached to port LPT1. In a LAN environment, the LPT1 device may be assigned to a server queue serviced by PSF/2. For example:

  ```
  net use lpt1 \\r22srv11\pr3820
  print config.sys /d:lpt1
  ```

  would result in the file CONFIG.SYS being directed to the PR3820 queue on the server named R22SRV11.

- **Using OS/2 COPY or TYPE**

  A favorite technique used by OS/2 users to print a file is to redirect the output of the COPY or TYPE command to a printer device. For example, the commands:

  ```
  copy config.sys > lpt1:
  type config.sys > lpt1:
  ```

  would both result in the file CONFIG.SYS being directed to the printer attached as LPT1. As with the DOS example, the output may be directed to an OS/2 PrintManager queue serviced by PSF/2, for example:

  ```
  net use lpt1 \\r22srv11\pr3820
  copy config.sys > lpt1:

           OR

  copy config.sys > \\r22srv11\pr3820
  ```

  would both result in file CONFIG.SYS being sent to the OS/2 PrintManager queue named PR3820 on the server named R22SRV11.

- **Assigning PSF/2 as the Default Printer**

  OS/2 PrintManager permits the user to define a default print queue which will receive all print requests not explicitly directed to a queue. For instance, if the user requests that a help screen be printed, it will be directed to the default printer. The default printer is defined using the ***Application Defaults*** selection on the ***Setup*** pulldown on the main OS/2 PrintManager panel.

Additional Considerations

- **User Notification**

  If the MESSENGER and NETPOPUP services are running, a message popup is sent to the requester when the printing is finished. Unfortunately, the message always indicates that the file has been deleted and did not print. This effect is caused by the fact that PSF/2 first selects the file from the PrintManager queue, puts it back in hold status, and then deletes it when the printing has been successfully completed. PSF/2 must do this because the OS/2 spooler lacks the facilities to permit proper recovery of the printing in the event of an error. Therefore, PSF/2 must retain a copy of the print file in the queue by using the hold technique. An unfortunate side effect of this is

that when PSF/2 finally deletes the job when the printing is complete, the OS2 PrintManager sends a message to the requester that the file has been deleted and did not print.

- **Banner Information**

  Without using the APRINT command, it is not possible for the user to alter any information that appears on the banner page. The LAN user ID from which the request comes will appear in the large block letters.

- **Accounting/Audit Information**

  PSF/2 provides an exit point for the extraction of auditing and accounting information. No standard code is provided for this exit. Customers may write their own routines to capture the information they deem useful.

Discussion of Data streams: Please see the definitions provided on page 2 for descriptions of the data stream types referenced in the following items.

- **Flat file**

  ASCII flat files generally print correctly. PSF/2 attempts to identify the type of input data file it receives and handle it appropriately. In the unlikely case of an ASCII file accidentally appearing to conform to valid AFPDS structures, it will be necessary to use the APRINT command and specify DATATYPE=ASCII.

- **Full AFPDS**

  It is not possible to print AFPDS files using the DOS PRINT command. The PRINT command does some character translation that invalidates AFPDS data.

  From an OS/2 command prompt, full AFPDS files print with all AFP function supported. If the file contains inline resources, they will be used. All resource types, with the exception of page definition, are supported inline.

  Note: It is not possible to mix AFPDS and non-AFPDS data in the same print file. When PSF/2 encounters the first data that is not valid AFPDS structures, it terminates the print job.

- **Presentation Manager Metafile**

  OS/2 Presentation Manager applications may generate Presentation Manager Metafiles. Often they contain graphics. It is not possible to print Presentation Manager Metafiles using the DOS PRINT command. The PRINT command does some character translation that invalidates the data. There are no special considerations when printing Presentation Manager Metafiles from an OS/2 command prompt.

- **QuietWriter ASCII data**

  Any ASCII print file generated using the QuietWriter3 level of printing escape sequences, or a subset thereof [2], will print with no special considerations. You must be aware, however, that this support is an emulation of QuietWriter function and does not support everything in exactly the same fashion as a real QuietWriter.

  PSF/2 attempts to identify the type of input data file it receives and handle it appropriately. In the unlikely case of an ASCII file accidentally appearing to

[2] ProPrinter ASCII is a subset of QuietWriter ASCII.

conform to valid AFPDS structures, it will be necessary to use the APRINT command and specify DATATYPE=ASCII.

8.6.2 Print Resource Migration

Migration of resources from one OS/2 to another OS/2 system is trivial.

If a telecommunications connection, for example a LAN connection, exists between systems, the resources can be copied by using OS/2 system commands. The resources in one system are also accessible from another system without copying. To be accessible by a PSF/2 program the resources have to be registered in the PSF/2 resource database by using the RLADD command.

Moving resources from one system to another by using a diskette or diskettes is also possible. In this case, the resources have to be added to the receiving system by using the RLADD command before they are accessible by PSF/2.

Appendix A. PSF/MVS Exits and MVS Sample Programs

This appendix documents the exits and utility programs that we used on the MVS platform to print our test cases.

All of the coding documented in this chapter is presented as *sample coding only*.

Be sure that you have read the information in “Special Notices” on page ix.

The following table serves as an index to the various routines.

Name	Language	Description	Page
AFPDSFIX	REXX	Routine that reconstructs AFPDS structured fields that have been “streamed” during BINARY File Transfer Protocol transport.	73
CRTRSC2	REXX	Routine that extracts AFP inline resources from an AFPDS file	76
APSUX04	Assembler	PSF/MVS Exit routine that operates in conjunction with a companion APSUX07 exit to provide full inline resource support for PSF/MVS.	77
APSUX07	Assembler	PSF/MVS Exit routine that operates in conjunction with a companion APSUX04 exit to provide full inline resource support for PSF/MVS.	85
ILRPACK	Assembler	Utility program that packs AFP resource objects inline with the print file.	94
LN2AFPDS	PL/I	Utility program that constructs an AFPDS output file from a line-data input file and a pagedef. This program is an updated version of the LINEAFP program available from Boulder on the MVSTOOLS disk.	119

A.1 AFPDSFIX Routine

In order to preserve all the code points within the data, AFPDS files being transported must use the BINARY translate tables. This causes the data to be transmitted without any translation of code points. It also causes the data to arrive at the receiving system as a continuous stream of bytes.

The data is not received with each AFPDS structured field record occupying its own variable length record, which is the format expected by PSF/MVS and PSF/VM. Instead, the records are formed by “streaming” the data into each record to the maximum defined record length, without regard for the lengths of the actual AFPDS structured fields. Several AFPDS structured fields may occupy the same record or, conversely, a single AFPDS structured field may span more than one physical record.

The purpose of the AFPDSFIX routine is to reconstruct the AFPDS data such that each AFPDS structured field occupies its own variable length record.

Although coded in REXX and executed under TSO/E, the AFPDSFIX routine was designed to run as a TSO batch job. The logic assumes the main datasets have been preallocated. The input file must be allocated to the SYSUT1 DDNAME and the output file to the SYSUT2 DDNAME.

© Copyright IBM Corp. 1994

A.1.1 REXX Coding

```
/* REXX              */
 /*                                                                  */
 /*------------------------------------------------------------------*/
 /*            ROUTINE TO COPY PRINT FILE TO MVS SPOOL               */
 /*                                                                  */
 /* This routine copies the print data pointed at by DD SYSUT1 to    */
 /* DD SYSUT2.  The logic assumes that the routine is being executed */
 /* in TSO batch and that the files have been preallocated in the    */
 /* startup JCL.  It is also assumed that all print output parameters*/
 /* have been specified in the startup JCL.                          */
 /*                                                                  */
 /* The routine examines the input file for AFPDS structured field   */
 /* records that may have been reformatted during a transmission     */
 /* from OS/2.  During such a transmission, the structured fields    */
 /* are treated as a stream of data and the original records are     */
 /* lost.  The transmitted file may have many structured fields in   */
 /* a single record, or an individual structured field may span      */
 /* multiple records.                                                */
 /*                                                                  */
 /* When properly formed AFPDS structured fields are detected they   */
 /* are written to the output as 1 structured field per output       */
 /* record.  Data not occuring within a structured field is written  */
 /* out according to the following rules:                            */
 /*                                                                  */
 /*    1. If no structured field introducer is present in the        */
 /*       record, the entire record is written unaltered.            */
 /*    2. If a structured field begins within the record, the data   */
 /*       preceding that structured field is written as a record.    */
 /*    3. Data falling between two valid structured fields is        */
 /*       written as a record.                                       */
 /*    4. Data following a valid structured field is written as a    */
 /*       record, with one exception.  If the structured field is    */
 /*       IPO, IPS, IMM, or IDM and the remainder of the record in   */
 /*       which it is found is blank, it is assumed to be a          */
 /*       valid structured field control record imbedded in a fixed  */
 /*       length record data file.  The trailing blanks are stripped */
 /*       and ignored.                                               */
 /*                                                                  */
 /* This logic has the following effects:                            */
 /*                                                                  */
 /*    1. Files with no structured field content are transcribed     */
 /*       verbatim.                                                  */
 /*                                                                  */
 /*    2. Files containing only structured fields are written out    */
 /*       with 1 structured field per record.                        */
 /*                                                                  */
 /*    3. Files with a mixture of structured field records and other */
 /*       data may or may not be reconstructed accurately.  Since    */
 /*       information about the original record lengths has been     */
 /*       lost, only AFPDS records can be accurately reconstructed.  */
 /*       If non-AFPDS data are isolated to their own records, the   */
 /*       reconstruction should be accurate.                         */
 /*                                                                  */
 /*------------------------------------------------------------------*/
 /*                                                                  */
/* REXX              */
/*                   */ say ">>AFPDSFIX Routine invoked to restructure AFPDS Records"
/*Prime nextrec buffer*/ nextrec = readrec()
/*Clear currec       */ currec  = ""
/*Str=IPO IPS IMM IDM */ chkstrng = x2c('d3afd840d3af5f40d3abcc40d3abca')
/*Init 1st char indic */ char1 = 0
/*                   */
/*Main execution loop */
/*                   */ Do Forever
/*Get an output rec   */    outrec1 = get_outrec()
/*If end of file, quit*/    if outrec1 = "*EOF*" then leave
/*Write output rec    */    Address TSO "EXECIO 1 DISKW SYSUT2 (STEM OUTREC)"
/*End main loop       */ end
/*                   */ ndit:
/*Close output file   */    Address TSO "EXECIO 0 DISKW SYSUT2 (FINIS)"
/*Close input file    */    Address TSO "EXECIO 0 DISKR SYSUT1 (FINIS)"
/*Scram               */    exit
 /*                                                                  */
 /*------------------------------------------------------------------*/
 /*                   Get Next Output Record                         */
 /* This routine isolates the next record to be written to the       */
 /* output file. CURREC contains the current data record, NEXTREC    */
 /* contains the next record from the input file.  When CURREC is    */
 /* fully processed, NEXTREC is moved to CURREC and a new record     */
 /* is read into NEXTREC from the input file.  The logic isolates    */
 /* the next output record using the rules described earler.  The    */
```

```
 /* isolated record is returned to the caller and stripped from      */
 /* CURREC.  If the isolated record is a valid IPO, IPS, IMM, or IDM */
 /* and the remainder of CURREC is blank, it is assumed the str fld  */
 /* is imbedded in a fixed length record, followed by trailing blanks*/
 /* The blanks are stripped and not treated as a following data rec. */
 /*------------------------------------------------------------------*/
 /*                                                                  */
/*                   */ Get_Outrec: procedure expose currec nextrec chkstrng char1
/*If currec empty,    */    if length(currec) < 1 then do
/* move in nextrec and*/       currec = nextrec
/* read next record   */       nextrec = readrec()
/*Ind 1st char of rec */       char1 = 1
/*                    */    end
/*Hit eof, quit       */    if currec = "*EOF*" then return currec
/*Start at position 0 */    candidate_pos = 0
/*Isolation loop      */    Do forever
/*Look for "!"        */       candidate_pos = pos("!",currec,candidate_pos+1)
/*If none             */       if candidate_pos = 0 then do
/* return whole record*/          outrec = currec
/* to caller          */          currec = ""
/*                    */          return outrec
/*                    */       end
/*Found a "!"         */       else do
/*Check for strfld rec*/          strlen = ver_strfld(currec||nextrec)
/*If not valid,       */          if strlen = -99 then do
/* bump position and  */             candidate_pos = candidate_pos + 1
/*Not at 1st char now */             char1 = 0
/* look for next "!"  */             iterate
/*                    */          end
/*Get strfld mneumonic*/          mneumonic = substr(currec,4,3)
/*ctlrec in rec pos 1?*/          if 0 < wordpos(mneumonic,chkstrng) & char1 then do
/* with len>currec len*/             if strlen > length(currec),
/* followed by linedat*/             &  substr(currec||nextrec,strlen+1,1) <> "!" then do
/* then blankpad rec  */                outrec = substr(currec,1,strlen)
/* clear currec       */                currec = ""
/* return strfld ctl  */                return outrec
/*                    */             end
/*                    */          end
/*                    */
/*Found good strfld   */
/*Must loop to read   */
/* all data in strfld */
/*                    */
/*Not at 1st char now */          char1 = 0
/*Get whole str field */          do while strlen > length(currec)
/* oops               */             if nextrec = "*EOF*" then do
/* bad str fld at end */                say "Found incomplete structured field record at",
/*                    */                    "end of input file"
/* save what we have  */                outrec = currec
/* blank currec       */                currec = ""
/* return incompl rec */                return outrec
/*                    */             end
/* append nextrec     */             currec = currec||nextrec
/* get another rec    */             nextrec = readrec()
/*end loop            */          end
/*Isolate strfld rec  */          strfld = substr(currec,1,strlen)
/*Update currec       */          currec = substr(currec,strlen+1)
/*Get strfld mneumonic*/          mneumonic = substr(strfld,4,3)
/*Str=IPO IPS IMM IDM */          chkstrng = x2c('d3afd840d3af5f40d3abcc40d3abca')
/*If rec one of these */          if 0 < wordpos(mneumonic,chkstrng),
/* and remainder blank*/           & strip(currec) = ""
/* assume fixed len   */          then currec = ""
/* str fld rec        */
/*Return to caller    */          return strfld
/*                    */       end
 /*                                                                  */
 /*------------------------------------------------------------------*/
 /*                   Read a Physical Record                         */
 /* This routine reads the next physical record from the input file  */
 /* and returns it to the caller.                                    */
 /*------------------------------------------------------------------*/
 /*                                                                  */
Readrec: procedure
/*                    */    Address TSO "EXECIO 1 DISKR SYSUT1 (STEM DISKREC)"
/*                    */    if rc > 0 then return "*EOF*"
/*                    */    return diskrec1
 /*                                                                  */
 /*------------------------------------------------------------------*/
 /*                   Verfify a Structured Field                     */
 /* This routine verifys that the data string passed as an argument  */
 /* conforms to the rules for a valid AFPDS structured field.  If    */
 /* the argument appears to be a structured field, the length of     */
 /* structured field is retrieved from the AFPDS length field,       */
```

```
 /* incremented by 1 to account for the "!" byte and returned to     */
 /* the caller. If the argument doesn't verify, -99 is returned.     */
 /*------------------------------------------------------------------*/
 /*                                                                  */
Ver_Strfld: procedure
/*                    */    parse arg 1 cc 2 len 4 flag1 5 flag2 6 .
/*1st char x'5a'?     */    if cc    <> "!" then return -99
/*4th char x'd3'?     */    if flag1 <> "L" then return -99
/*5th char in list?   */    if 0 <> verify(flag2,x2c('aaabacaeafa2a6a7a8a9b1b6ee8c')) then return -99
/*return str fld leng */    return 1+c2d(len)
```

A.2 Routine to Extract AFP Inline Resources

This REXX EXEC extracts AFP inline resources from an AFPDS file. It can be used, for example, to extract resources from an AFPDS file created by OS/400 Advanced Function Printing Utility (AFPU).

A.2.1 REXX Coding

```
/* REXX EXEC TO EXTRACT RESOURCES FROM AN AFPDS FILE */                00010000
   SAY 'GIVE THE NAME OF THE INPUT FILE'                               00020002
   SAY '(FULLY QUALIFIED NAME WITHOUT APOSTROPHES, PLEASE)'            00021002
PARSE UPPER PULL AFPNIMI                                               00030000
   SAY 'GIVE THE NAME OF THE OUTPUT FILE (LIBRARY)'                    00040002
   SAY '(FULLY QUALIFIED NAME WITHOUT APOSTROPHES, PLEASE)'            00041002
PARSE UPPER PULL AFPOVER                                               00050000
DATASEI = 'DA(''' || AFPNIMI || ''')'                                  00060002
   SAY 'EXTRACTING RESOURCES FROM' AFPNIMI                             00070002
   SAY 'AND PLACING THEM TO THE LIBRARY ' AFPOVER                      00080002
"ALLOC DD(AFPDS)" DATASEI " SHR REUSE"                                 00100000
  RETU=RC                                                              00110003
  IF RETU<>0 THEN DO                                                   00120003
   SAY 'INPUT DATASET NOT FOUND'                                       00121003
   EXIT                                                                00122003
  END                                                                  00123003
  "EXECIO * DISKR AFPDS (FINIS STEM RIVI."                             00130003
    COUNT=RIVI.0                                                       00140003
    SAY 'RECORD COUNT IN INPUT DATASET' COUNT                          00150003
   DO I = 1 TO COUNT                                                   00151003
     IF SUBSTR(RIVI.I,4,3)=X2C('D3A9C6') THEN DO                       00153003
      SAY 'ERG ENCOUNTERED '                                           00154003
      EXIT                                                             00155003
      END                                                              00156003
     IF SUBSTR(RIVI.I,4,3)=X2C('D3A8C6') THEN LEAVE                    00157003
      END                                                              00158003
   IF I > COUNT THEN DO                                                00159103
    SAY 'BRG NOT FOUND'                                                00159203
    EXIT                                                               00159303
    END                                                                00159403
   IALKU=I                                                             00160003
    DO FOREVER                                                         00170003
    DO J = IALKU TO COUNT                                              00190003
    IF SUBSTR(RIVI.J,4,3)=X2C('D3A9C6') THEN DO                        00340003
     EXIT                                                              00340103
     END                                                               00340203
     IF SUBSTR(RIVI.J,4,3)=X2C('D3A8CE') THEN LEAVE                    00341003
    END                                                                00350003
    IF J > COUNT THEN DO                                               00370003
     SAY 'BR NOT FOUND'                                                00380003
     EXIT                                                              00390003
     END                                                               00400003
    SAY 'RESOURCE STARTING FROM RECORD ' J                             00401003
    RESNAME = STRIP(SUBSTR(RIVI.J,10,8),'T')                           00410003
    SAY 'RESOURCE NAME ' RESNAME                                       00420003
       DATASEO = 'DA(''' || AFPOVER || '(' || RESNAME || ')'')'        00500003
   SAY 'STORING TO MEMBER' DATASEO                                     00510003
   "ALLOC DD(AFPOU)" DATASEO " SHR REUSE"                              00520003
    RETU=RC                                                            00530003
    IF RETU<>0 THEN DO                                                 00540003
    SAY 'ALLOCATION OF OUTPUT DATASET FAILED'                          00541003
     EXIT                                                              00543003
     END                                                               00544003
   DO  I= J+1 TO COUNT                                                 00560002
    IF SUBSTR(RIVI.I,4,3)=X2C('D3A9CE') THEN LEAVE                     00590003
    RDWLEN=X2D(C2X(SUBSTR(RIVI.I,2,2)),4)                              00590104
   RECLEN=RDWLEN+1                                                     00590304
   TEMP = SUBSTR(RIVI.I,1,RECLEN,' ')                                  00590404
   QUEUE TEMP                                                          00591004
```

```
   END                                                          00600003
IF I > COUNT THEN DO                                            00620002
 SAY 'ER NOT FOUND'                                             00630000
 EXIT                                                           00640000
END                                                             00650000
ELSE DO                                                         00660002
   SAY 'END OF RESOURCE FOUND IN RECORD ' I                     00661003
   QUEUE ""                                                     00670000
   "EXECIO * DISKW AFPOU (FINIS"                                00680000
   IALKU=I+1                                                    00700002
   END                                                          00710002
   END                                                          00720002
```

A.3 PSF/MVS Inline Resource Exit APSUX04

AFPDS defines a structure known as an Inline Resource Group which may be present at the beginning of any print file. This structure may contain resource objects to be used in printing the file. Objects found within an inline resource group will be used in preference to identically named objects in the normal system libraries. This capability makes it possible to build a completely self-contained print file that does not require any resources to be installed on the driving system before it can be printed.

PSF/VM Version 2 Release 1 and OS/400 Version 2 permit any type of resource to be defined in an inline resource group. PSF/2 Version 1 Release 1 on the OS/2 platform permits all resources except the pagedef, which is not a valid object on OS/2. PSF/MVS Version 2 Release 1 and PSF/VSE Version 2 Release 1 permit only pagedefs and formdefs to be defined inline.

The PSF/MVS exit coding in this section, in concert with its companion APSUX07 exit documented in A.4, "PSF/MVS Inline Resource Exit APSUX07" on page 85, provides a facsimile of full inline resource support for PSF/MVS. There are a number of restrictions and caveats which are explained in the comments that appear at the beginning of the coding. Please read them *carefully*.

A.3.1 Sample Assembler Code

```
*************** APSUX04 PSF LOGICAL RECORDS EXIT*********************
*         <<< MVS INLINE RESOURCE PROCESSING EXIT >>>               *
*                                                                   *
*  The exit coding in this module and the corresponding APSUX07     *
*  module provides full inline resource support by capturing inline *
*  objects not understood by PSF/MVS (as of PSF/MVS V2R1 that means *
*  overlays, page segments, and font objects) and writing them to a *
*  dataset that is part of PSF's concatenation of resource libraries.*
*  Thus, when PSF references the object, it is available in the     *
*  normal resource library structure.                               *
*                                                                   *
*  Correct operation of this exit coding requires that DD statements *
*  be added to the PSF startup JCL.  See the comments in the APXUX07 *
*  coding for details on the required JCL.                          *
*                                                                   *
*  APSUX07 is responsible for setting up the control blocks, opening *
*  files, deleting PDS members, and other housekeeping functions    *
*  required by this exit.  See the APSUX07 code for details.        *
*                                                                   *
*  This exit is responsible for examining the records being read    *
*  from the JES spool, detecting the presence of an inline resource *
*  object of a type not understood by PSF/MVS, writing such objects *
*  to the appropriate temporary resource file, and deleting them    *
*  from the PSF datastream.  Inline pagedefs and formdefs are passed *
*  through to PSF unaltered.                                        *
*                                                                   *
*                                                                   *
```

```
*  BASIC LOGIC:                                                        *
*                                                                      *
*    (Remember:  this exit sees each record as it comes off the        *
*                spool, one record per exit call)                      *
*                                                                      *
*    1. Locate workarea created by APSUXO7.                            *
*                                                                      *
*    2. Test current status.  If we are currently diverting an         *
*       inline resource to a temporary dataset, go to step 5.          *
*                                                                      *
*    3. If we are not currently within an inline resource that is      *
*       being diverted, then check to see if current record is a       *
*       Begin Resource (BR) record for an inline resource requiring    *
*       diversion.  If not, record is passed to PSF; exit execution    *
*       complete.                                                      *
*                                                                      *
*    4. If current record is a BR for a type requiring diversion,      *
*       then:                                                          *
*                                                                      *
*         a) ILRFLAGS is set to indicate a resource is currently       *
*            being diverted.                                           *
*                                                                      *
*         b) The write buffer area WRTBUFFR is initialized for the     *
*            correct dataset by storing the address of the DCB for     *
*            the type of object being diverted, the block size of      *
*            that file, resetting the block descriptor word to         *
*            indicate no records in the block (yet) and resetting the*
*            next record pointer to point to the start of the buffer.*
*                                                                      *
*         c) Then the next available BLDL entry is updated with the    *
*            object name found in the Begin Resource (BR) record and   *
*            the address of the DCB to which it is being written.      *
*                                                                      *
*    5. If we are currently diverting a resource, then:                *
*                                                                      *
*         a) Check to see if current record is an End Resource (ER)    *
*            record.  If so, go to step 6.                             *
*                                                                      *
*         b) Check to see if current record will fit into remaining    *
*            space in buffer, if so move it into buffer, update        *
*            appropriate control fields, and delete record from PSF    *
*            datastream.                                               *
*                                                                      *
*         c) If current record length + used buffer space > block      *
*            size of file, then write the current buffer to DASD,      *
*            reinitialize pointers to indicate buffer is empty again   *
*            and go back to beginning of Step 5.                       *
*                                                                      *
*    6. If current record is an End Resource (ER) then:                *
*                                                                      *
*         a) Write out any partially-filled IO buffer and reset        *
*            pointers to indicate buffer is empty.                     *
*                                                                      *
*         b) Issue a STOW to update PDS directory for member just      *
*            completed.                                                *
*                                                                      *
*         c) Reset ILRFLAGS to indicate no resource is currently       *
*            being diverted.                                           *
*                                                                      *
*                                                                      *
*                                                                      *
*               ########################################             *
*               #                                      #             *
*               #  WARNING * WARNING * WARNING * WARNING  #          *
*               #               RESTRICTIONS           #             *
*               #                                      #             *
```

```
*              ########################################          *
*                                                                *
*  This exit has the following restrictions:                     *
*                                                                *
*     -  There a number of restrictions and caveats associated with  *
*        the use of the functions provided by this exit and its      *
*        companion APSUX07 module.  They are documented in the       *
*        APSUX07 coding.                                             *
*                                                                *
*     -  This exit depends upon pointers and data areas being set up *
*        by corresponding code in APSUX07.  If this exit is executed *
*        in the absence of that APSUX07 coding, an ABEND (most       *
*        likely a 0C4) will occur.                                   *
*                                                                *
*     -  The above logic was designed for simplicity and makes no    *
*        attempt to analyze the incoming datastream for errors.  If  *
*        the user builds an invalid resource group or an invalid     *
*        resource within a group, the error will not be detected     *
*        until the object is first examined by PSF.  In most cases,  *
*        this will not constitute a problem.  PSF will issue the     *
*        same error that would have been appropriate if the exit     *
*        were not present.  However, in the case where a Begin       *
*        Resource record is detected, thereby causing the resource   *
*        to be diverted into the temporary datasets, and no          *
*        subsequent End Resource record is encountered by this exit, *
*        then the entire print dataset will be written into the      *
*        temporary dataset.  If the dataset is large this may cause  *
*        an E37 ABEND in the exit (and therefore, in PSF).  If that  *
*        does not happen, the error may be difficult to detect       *
*        because PSF will simply see a null print dataset, and       *
*        therefore no error messages will be issued.                 *
*                                                                *
*     -  This exit uses a number of MVS system service macros.  The  *
*        coding as written does not do error checking or result      *
*        validation on return from these calls.  Therefore, an IO    *
*        error or other recoverable failure could result in an       *
*        ABEND of the PSF address space.                             *
*                                                                *
*     -  This exit will run ONLY AMODE 24, RMODE 24.                 *
*                                                                *
*     -  If PSF is not marked non-swappable in the PPT, unpredictable*
*        abends may occur.                                           *
*                                                                *
*  The basic logic and environment for PSF exits are described in    *
*  the PSF System Programming Guide (S544-3672).                     *
*                                                                *
*                                                                *
******************************************************************
APSUX04  START 0
         TITLE 'DSECT - XTP'
         APSGEXTP LIST=YES
         TITLE 'DSECT - ECA'
         APSUECA LIST=YES
         TITLE 'INLINE RESOURCE HANDLING EXIT (APSUX04)'
APSUX04  CSECT ,
APSUX04  AMODE 24
APSUX04  RMODE 24
         USING *,R15
         B     START
         DC    AL1(16)                 LENGTH OF FOLLOWING FIELDS
         DC    CL8'APSUX04 '           NAME OF THIS ROUTINE
         DC    CL8'&SYSDATE'           DATE OF THIS ASSEMBLY
         DC    CL16'SOURCEM: V2X4ILR'
         DROP  R15
START    DS    0H
         STM   R14,R12,R12(R13)        SAVE CALLERS REGISTERS
```

```
         LR    BASEREG,R15             SWITCH BASE REGISTER
         USING APSUX04,BASEREG
         USING APSGEXTP,XTPPTR
         USING APSUECA,ECAPTR
         L     XTPPTR,0(,R1)           LOAD ADDRESS OF APSGEXTP
         L     ECAPTR,XTPECAP          LOAD ADDRESS OF APSUECA
         LR    R2,R13                  LOAD ADDRESS OF CALLERS SAVE
         LA    R13,ECAUSAVE            ADDRESS OF APSUX04 SAVE AREA
         ST    R2,4(,R13)              SAVE CALLERS SAVE AREA ADDRESS
         ST    R13,8(,R2)              SAVE APSUX04 SAVE AREA ADDRESS
         MVI   XTPPIND,XTPWRT          ENSURE DEFAULT IS TO WRITE
*                                      ORIGINAL RECORD
*-------------------------------------------------------------------*
*    Find ILRCOMM area at end of ECAWKBUF and load pointer to ILRWORK *
*-------------------------------------------------------------------*
         L     ILRPTR,ECALEN           ILRPTR = LEN OF ENTIRE ECA       00107516
         LA    R3,ILRCOMML             R3 = LEN OF ILRCOMM AREA         00107616
         SLR   ILRPTR,R3               ILRPTR = ECALEN-LEN OF ILRCOMM   00107716
         LA    ILRPTR,APSUECA(ILRPTR)  ILRPTR -> ILRCOMM FIELD          00107816
         L     ILRPTR,0(ILRPTR)        ILRPTR -> ILR WORK AREA
         USING ILRWORK,ILRPTR          ADDRESSABILITY TO ILRWORK AREA
         L     RECPTR,XTPRECP          RECPTR -> INPUT RECORD
         USING BRREC,RECPTR            ESTABLISH ADDRESSABILITY TO REC
*
*-------------------------------------------------------------------*
*           Determine current status and branch to proper handler    *
*-------------------------------------------------------------------*
         TM    ILRFLAGS,OVSTARTD+PSSTARTD+FOSTARTD   IS AN OVLY, PSEG,
*                                                  OR FONT IN PROCESS?
         BNZ   GODIVERT                IF YES, GO DIVERT IT TO TEMP DS
*
* ------------------------------------------------------------------*
*      At this point we know we are not currently processing an     *
*      inline resource that is being diverted to a file.  Check to  *
*      see if current record is a Begin Resource, if not give it to *
*      PSF.                                                         *
* ------------------------------------------------------------------*
         CLC   INOPCOD,BROC            IS INPUT A BEGIN RESOURCE REC?
         BNE   GETOUT                  IF NOT, GIVE IT TO PSF
*
* ------------------------------------------------------------------*
*   We found a Begin Resource record. Determine the type of resource *
*   and whether PSF can handle it or it gets diverted to the         *
*   temporary resource library.  ILRWORK contains a list of words,   *
*   one for each type of object that has to be diverted.  The        *
*   high-order byte of the word contains the flag that will be found *
*   in the BR record for that type of resource.  The low order 3     *
*   bytes contain the address of the DCB for the file to which that  *
*   object type is to be written.                                    *
* ------------------------------------------------------------------*
         LA    R2,4                    SET INCREMENT
         LA    R3,NDOBJLST             INITIALIZE END POINTER
         LA    R4,STOBJLST             INITIALIZE START POINTER
BROBJLP  EQU   *
         CLC   BROBJTYP,0(R4)          CHECK BR OBJECT TYPE
         BE    BRGOTOBJ                FOUND IT, GO PROCESS
         BXLE  R4,R2,BROBJLP           LOOP TO NEXT LIST ENTRY
         B     GETOUT                  NOT IN LIST, PSF GETS IT
BRGOTOBJ EQU   *
         L     R5,0(R4)                R5 -> DCB FOR OUTPUT FILE
         LA    R5,0(R5)                CLEAR FLAG FROM HIGH ORDER
         MVI   ILRFLAGS,X'0F'          INDICATE DIVERSION UNDERWAY
*
*-------------------------------------------------------------------*
*    Initialize IO control blocks for objects not going to PSF      *
*    (this routine assumes the DCB address for the file to be       *
```

```
*     written is contained in R5)                                  *
*-------------------------------------------------------------------*
*
INITIO   EQU   *
         USING IHADCB,R5              ESTABLISH ADDRESSABILITY TO DCB
         L     R2,WRTBUFAD            R2 -> WRITE BUFFER AREA
         USING WRTBUFFR,R2            ADDRESSABILITY TO WRITE BUFFER
         ST    R5,WRTBDCB             SAVE DCB ADDR FOR WRITE ROUTINE
         MVC   WRTBBDW,CONS4          INITIALIZE BLOCK DESC WORD
         LA    R3,WRTBRDW             R3 -> 1ST AVAIL REC LOCATION
         ST    R3,WRTBNEXT            SAVE FOR WRITE ROUTINE
         LH    R3,DCBBLKSI            R3 = MAX BLOCK SIZE
         ST    R3,WRTBBLKS            SAVE FOR WRITE ROUTINE
         DROP  R2                     DROP ADDRESSABILITY TO WRTBUFFR
         L     R2,BLDLLAST            R2 -> LAST USED BLDL ENTRY
         L     R3,BLDLLEN             LENGTH OF BLDL ENTRIES
         LA    R2,0(R3,R2)            R2 -> NEW BLDL ENTRY
         USING BLDLNTRY,R2            ESTABLISH ADDRESSABILITY TO BLDL
         ST    R5,BLDLDCB             SAVE DCB ADDRESS IN BLDL NTRY
         MVC   BLDLMNAM,BRRNAME       MOVE IN MEMBER NAME OF OBJECT
         SLR   R4,R4                  CLEAR R4
         ST    R4,BLDLFLGS            SET BLDL FLAG WORD TO ZEROS
         ST    R2,BLDLLAST            SAVE ADDR OF THIS BLDL ENTRY
         MVI   XTPPIND,XTPSKP         DELETE THIS BR REC FROM PSF
         B     GETOUT                 NOW WE GO
         DROP  R2                     DROP BLDL ADDRESSABILITY
*
*
**********************************************************************
*   This routine handles records within a non-PSF inline resource. It *
*   writes all records to the DCB addr contained in WRTBDCB field.    *
*   It looks for the End Resource marker and terminates the member     *
*   with a STOW when it is found.                                      *
**********************************************************************
*
GODIVERT EQU   *
         CLC   INOPCOD,EROC           IS INCOMING REC AN END-RESOURCE?
         BE    ERFILES                IF YES, GO END RESOURCE
         L     R1,WRTBUFAD            R1 -> OUTPUT BUFFER AREA
         USING WRTBUFFR,R1            ESTABLISH ADDRESSABILITY
         LH    R2,WRTBBDW             R2 = CURRENT BLOCK LENGTH
         LA    R2,4(R2)               BUMP IT UP 4 BYTES (FOR RDW)
         A     R2,XTPRECL             ADD LENGTH OF NEW RECORD
         C     R2,WRTBBLKS            WILL NEW RECORD FIT IN CURR BLK?
         BH    WRITEBLK               IF NOT, GO WRITE CURRENT BLOCK
         STH   R2,WRTBBDW             UPDATE BDW FOR NEW RECORD
         L     R2,WRTBNEXT            R2 -> NEXT RECORD LOCATION
         SLR   R3,R3                  CLEAR R3
         ST    R3,0(R2)               INITIALIZE RDW FOR NEW RECORD
         L     R5,XTPRECL             R5 = LENGTH OF NEW RECORD
         LA    R3,4(R5)               R3 = LEN NEW REC + RDW
         STH   R3,0(R2)               STORE NEW RDW
         LR    R3,R5                  R3 = LENGTH OF NEW RECORD
         LR    R4,RECPTR              R4 -> NEW RECORD FROM SPOOL
         LA    R2,4(R2)               R2 -> FIRST TARGET DATA BYTE
         MVCL  R2,R4                  MOVE RECORD TO OUTPUT BUFFER
         ST    R2,WRTBNEXT            SAVE NEXT AVAIL REC BYTE POINTER
         MVI   XTPPIND,XTPSKP         TELL PSF TO FORGET THIS RECORD
         B     GETOUT                 NOW WE'RE DONE
         DROP  R1                     DROP ADDRESSABILITY TO WRTBUFFR
*
*-------------------------------------------------------------------*
*   This routine writes a completed block to DASD and reinitializes  *
*   the buffer area.                                                 *
*-------------------------------------------------------------------*
WRITEBLK EQU   *
```

```
         LR    R2,R1                   R2 -> WRITE BUFFER AREA
         USING WRTBUFFR,R2             ESTABLISH NEW ADDRESSABILITY
         L     R3,WRTLSTAD             R3 -> LIST FORM WRITE MACRO
         L     R4,WRTBDCB              R4 -> DCB TO WRITE
         LA    R5,WRTBBDW              R5 -> BLOCK TO WRITE
         WRITE (R3),SF,(R4),(R5),MF=E  WRITE THE CURRENT BLOCK
         CHECK (R3)                    WAIT FOR IO TO COMPLETE
         MVC   WRTBBDW,CONS4           INDICATE AN EMPTY BUFFER
         LA    R3,WRTBRDW              R3 -> 1ST AVAIL REC LOCATION
         ST    R3,WRTBNEXT             SAVE FOR WRITE ROUTINE
         B     GODIVERT                LOOP BACK TO PROCESS CURR REC
         DROP  R2
*
*--------------------------------------------------------------------*
*   This routine handles the End-Resource condition.  It writes any  *
*   incomplete buffers and issues a STOW for the member.  Then it    *
*   turns off the object-in-process flags.                           *
*--------------------------------------------------------------------*
ERFILES  EQU   *
         L     R2,WRTBUFAD             R1 -> WRITE BUFFER AREA
         USING WRTBUFFR,R2             ESTABLISH NEW ADDRESSABILITY
         CLC   WRTBBDW,CONS4           ARE THERE UNWRITTEN RECORDS?
         BE    ERFILES1                BR IF NONE
         L     R3,WRTLSTAD             R3 -> LIST FORM WRITE MACRO
         L     R4,WRTBDCB              R4 -> DCB TO WRITE
         LA    R5,WRTBBDW              R5 -> BLOCK TO WRITE
         WRITE (R3),SF,(R4),(R5),MF=E  WRITE THE CURRENT BLOCK
         CHECK (R3)                    WAIT FOR IO TO COMPLETE
         MVC   WRTBBDW,CONS4           INDICATE AN EMPTY BUFFER
ERFILES1 EQU   *
         L     R3,BLDLLAST             R3 -> CURRENT BLDL ENTRY
         L     R4,WRTBDCB              R4 -> DCB FOR STOW
         STOW  (R4),(R3),R             UPDATE THE DIRECTORY
         MVI   ILRFLAGS,0              CLEAR IN-PROCESS FLAGS
         MVI   XTPPIND,XTPSKP          DON'T GIVE RECORD TO PSF
         B     GETOUT                  TIME TO GO
         DROP  R2
*
**********************************************************************
*                  EXIT ROUTINE                                      *
**********************************************************************
*
GETOUT   EQU   *
         SLR   R15,R15                 PSF EXPECTS ZERO RETURN CODE
         L     R13,4(,R13)             RESTORE CALLERS SAVE AREA ADDR.
         L     R14,12(,R13)            RESTORE CALLERS RETURN ADDRESS
         LM    R0,R12,20(R13)          RESTORE CALLERS REGISTERS
         BR    R14                     RETURN TO CALLER
         SPACE 2
*
**********************************************************************
*                     REGISTER AND OTHER EQUATES
**********************************************************************
R0       EQU   0                       SYMBOL FOR GP REG 0
R1       EQU   1                       SYMBOL FOR GP REG 1
R2       EQU   2                       SYMBOL FOR GP REG 2
R3       EQU   3                       SYMBOL FOR GP REG 3
R4       EQU   4                       SYMBOL FOR GP REG 4
R5       EQU   5                       SYMBOL FOR GP REG 5
R6       EQU   6                       SYMBOL FOR GP REG 6
R7       EQU   7                       SYMBOL FOR GP REG 7
R8       EQU   8                       SYMBOL FOR GP REG 8
R9       EQU   9                       SYMBOL FOR GP REG 9
R10      EQU   10                      SYMBOL FOR GP REG 10
R11      EQU   11                      SYMBOL FOR GP REG 11
R12      EQU   12                      SYMBOL FOR GP REG 12
```

```
R13      EQU   13                    SYMBOL FOR GP REG 13
R14      EQU   14                    SYMBOL FOR GP REG 14
R15      EQU   15                    SYMBOL FOR GP REG 15
ILRPTR   EQU   R6                    POINTER TO ILR WORK AREA
XTPPTR   EQU   R7                    POINTER TO APSGEXTP
ECAPTR   EQU   R8                    POINTER TO APSUECA
RECPTR   EQU   R9                    POINTER TO INPUT RECORD
BASEREG  EQU   R12                   BASE REGISTER
*
*********************************************************************
*                   DATA AREAS
*********************************************************************
CONS4    DC    H'4',H'0'             CONSTANT 4
*
*                                                                   *
*********************************************************************
*      AFPDS OPCODES                                                *
*********************************************************************
*                                                                   *
BROC     DC    XL3'D3A8CE'
EROC     DC    XL3'D3A9CE'
*                                                                   *
*********************************************************************
*    WTL LIST FORM FOR USE IN TRACING EXECUTION                     *
*                                                                   *
*********************************************************************
WTLLIST  WTL   'APSUX04:                                           ',*
               MF=L
WTLLISTL EQU   *-WTLLIST
*********************************************************************
*                                                                   *
*********************************************************************
*      DSECT TO MAP LIST FORM WTL BUILT WITH WTL MF=(L)             *
*********************************************************************
WTLDSECT DSECT
         DS    F
         DS    CL9
WTLTEXT  DS    CL47
*                                                                   *
*********************************************************************
* DSECT FOR ILR COMM AREA AT END OF UECA ECAWKBUF FIELD             *
*********************************************************************
ILRCOMM  DSECT
ILRPARMS DS    OF          POINTER TO GETMAINED PARAMETER AREA
ILRWKPTR DS    AL4         ADDRESS OF GETMAINED WORK AREA
ILRCOMML EQU   *-ILRCOMM   LENGTH OF ILR COMM AREA
*
*
*********************************************************************
*          ILRWORK  CONTROL BLOCKS DSECT STARTS HERE
*********************************************************************
*
ILRWORK  DSECT
ILRFLAGS DS    AL1         CONTROL FLAGS
OVSTARTD EQU   B'00000001'  OVERLAY BEING PROCESSED (BR FOR OVLY FOUND)
PSSTARTD EQU   B'00000010'  PSEG BEING PROCESSED (BR FOR PSEG FOUND)
FOSTARTD EQU   B'00000100'  FONT BEING PROCESSED (BR FOR FONT FOUND)
         DS    AL3
WRTBUFAD DS    AL4                   POINTER TO OUTPUT BUFFER
OPENLAD  DS    AL4                   POINTER TO OPEN PARM LIST
*-------------------Start Of Object DCB Table----------------------*
STOBJLST EQU   *                     ADDR OF FIRST ENTRY IN OBJECT LIST
OVLDCBAD DS    AL4                   POINTER TO OVERLAY DCB
PSGDCBAD DS    AL4                   POINTER TO PSEG DCB
FCSDCBAD DS    AL4                   POINTER TO FONT DCB (CHAR SETS)
FCPDCBAD DS    AL4                   POINTER TO FONT DCB (CODE PAGES)
```

```
NDOBJLST EQU   *                    ADDR OF LAST ENTRY IN OBJECT LIST
FCFDCBAD DS    AL4                  POINTER TO FONT DCB (CODED FONTS)
*---------------------End Of Object DCB Table----------------------*
WRTLSTAD DS    AL4                  POINTER TO WRITE PARM LIST
BLDLLAST DS    AL4                  POINTER TO LAST BLDL ENTRY
BLDLLEN  DS    AL4                  LENGTH OF EACH BLDL ENTRY
         DS    CL100                BUNCHES MORE STUFF THAT IS
*                                      POINTED AT BY ABOVE ADDRESSES
*************************************************************************
*          BLDL ENTRY DSECT STARTS HERE
*************************************************************************
*
BLDLNTRY DSECT              DSECT FOR ENTRIES
BLDLENT  DS    0CL16      BLDL LIST ENTRY
BLDLMNAM DS    CL8        MEMBER NAME
BLDLFLGS DS    XL4        FLAG WORD
BLDLDCB  DS    AL4        ADDRESS OF DCB THAT MEMBER WAS WRITTEN TO
*                                                                       *
*                                                                       *
*************************************************************************
*          WRITE BUFFER DSECT STARTS HERE
*  This DSECT is used by the GODIVERT and WRITEBLK routines to do
*  the file IO to divert an object into a temporary dataset.  The
*  first 3 words are control information and the rest of the area
*  is the buffer into which the output block is being built.
*  The values for the DCB address, next record address, and current
*  blocksize are set up by the INITIO routine when the BR record
*  is first encountered in the incoming datastream.
*************************************************************************
*
WRTBUFFR DSECT              DSECT FOR THE WRITE OUTPUT BUFFER
WRTBDCB  DS    AL4        POINTER TO DCB BEING WRITTEN
WRTBNEXT DS    AL4        POINTER TO NEXT AVAILABLE RECORD LOCATION
WRTBBLKS DS    AL4        MAX BLOCK SIZE FOR FILE BEING WRITTEN
WRTBBDW  DS    CL4        BLOCK DESCRIPTOR WORD
WRTBRDW  DS    CL4        FIRST RECORD DESCRIPTOR WORD
*                                                                       *
*************************************************************************
*      AFPDS RECORDS DSECTS                                             *
*************************************************************************
*                                                                       *
BRREC    DSECT         BEGIN RESOURCE RECORD
INCC     DS    CL1       X'5A'
INLEN    DS    CL2
INOPCOD  DS    CL3
INSEQ    DS    CL3
BRRNAME  DS    CL8       RESOURCE NAME
         DS    CL4       CONSTANT FLAGS
BROBJTYP DS    CL1       OBJECT TYPE FLAGS
BRCSET   EQU   X'40'     CHARACTER SET FLAG
BRCPAGE  EQU   X'41'     CODE PAGE FLAG
BRCFONT  EQU   X'42'     CODED FONT FLAG
BRPSEG   EQU   X'FB'     PAGE SEGMENT FLAG
BROVLY   EQU   X'FC'     OVERLAY FLAG
BRPDEF   EQU   X'FD'     PAGEDEF FLAG
BRFDEF   EQU   X'FE'     FORMDEF FLAG
*                                                                       *
*************************************************************************
*      DCB DSECT EXPANSION                                              *
*************************************************************************
*                                                                       *
         DCBD   DSORG=(PO),DEVD=(DA)
         END    APSUX04
/*
```

A.4 PSF/MVS Inline Resource Exit APSUX07

AFPDS defines a structure known as an Inline Resource Group which may be present at the beginning of any print file. This structure may contain resource objects to be used in printing the file. Objects found within an inline resource group will be used in preference to identically named objects in the normal system libraries. This capability makes it possible to build a completely self-contained print file that does not require any resources to be installed on the driving system before it can be printed.

PSF/VM Version 2 Release 1 and OS/400 Version 2 permit any type of resource to be defined in an inline resource group. PSF/2 Version 1 Release 1 on the OS/2 platform permits all resources except the pagedef, which is not a valid object on OS/2. PSF/MVS Version 2 Release 1 and PSF/VSE Version 2 Release 1 permit only pagedefs and formdefs to be defined inline.

The PSF/MVS exit coding in this section, in concert with its companion APSUX04 exit documented in A.3, “PSF/MVS Inline Resource Exit APSUX04” on page 77, provides a facsimile of full inline resource support for PSF/MVS. There are a number of restrictions and caveats which are explained in the comments that appear at the beginning of the coding. Please read them *carefully*.

A.4.1 Sample Assembler Code

```
********************* APSUX07 PSF RESOURCE EXIT***********************
*         <<< MVS INLINE RESOURCE PROCESSING EXIT >>>                *
*                                                                    *
*  The exit coding in this module and the corresponding APSUX04      *
*  module provide full inline resource support by capturing inline   *
*  objects not understood by PSF/MVS (as of PSF/MVS V2R1 that means  *
*  overlays, page segments, and font objects) and writing them to a  *
*  dataset that is part of PSF's concatenation of resource libraries.*
*  Thus, when PSF references the object, it is available in the      *
*  normal resource library structure.                                *
*                                                                    *
*  Correct operation of this exit coding requires that DD statements *
*  be added to the PSF startup JCL.  Three DD statements must be     *
*  added to reference the temporary resource libraries to which      *
*  inline resources will be written.  They must have the following   *
*  DDNAMES:                                                          *
*           INLNOVLY:  Points to the temporary overlay library       *
*           INLNPSEG:  Points to the temporary page segment library  *
*           INLNFONT:  Points to the temporary font library          *
*                                                                    *
*  In addition, the SAME datasets that are referenced by these 3     *
*  DDNAMES must also appear FIRST in the concatenation for their     *
*  respective PSF resource libraries.  Thus, if PSEG01 is the DDNAME *
*  that defines PSF's page segment library, the library referenced   *
*  by DDNAME INLNPSEG must appear first in the concatenation of      *
*  datasets referenced by DDNAME PSEG01.  This means that these      *
*  temporary datasets must have a format compatible with PSF resource*
*  libraries and have a block size that is equal or larger than any  *
*  of the libraries following in the concatenation.  A sample of the *
*  necessary JCL coding is shown below:                              *
*                                                                    *
*  //INLNOVLY DD  DSN=&&OVERLIB,DISP=(,PASS),UNIT=SYSDA,            *
*  //             SPACE=(CYL,(10,10,50)),                           *
*  //             DCB=(RECFM=VBM,LRECL=8205,BLKSIZE=8209)           *
*  //INLNPSEG DD  DSN=&&PSEGLIB,DISP=(,PASS),UNIT=SYSDA,            *
*  //             SPACE=(CYL,(10,10,50)),                           *
*  //             DCB=(RECFM=VBM,LRECL=8205,BLKSIZE=8209)           *
*  //INLNFONT DD  DSN=&&FONTLIB,DISP=(,PASS),UNIT=SYSDA,            *
```

```
*  //             SPACE=(CYL,(10,10,50)),                          *
*  //             DCB=(RECFM=VBM,LRECL=8205,BLKSIZE=8209)          *
*  //OLAY01   DD  DSN=&&OVERLIB,DISP=SHR,                          *
*  //             UNIT=SYSDA,VOL=REF=*.INLNOVLY                    *
*  //         DD  DSN=AFPDEMO.OVERLIB,DISP=SHR                     *
*  //         DD  DSN=SYS1.OVERLIB,DISP=SHR                        *
*  //PSEG01   DD  DSN=&&PSEGLIB,DISP=SHR,                          *
*  //             UNIT=SYSDA,VOL=REF=*.INLNPSEG                    *
*  //         DD  DSN=AFPDEMO.PSEGLIB,DISP=SHR                     *
*  //         DD  DSN=SYS1.PSEGLIB,DISP=SHR                        *
*  //FONT01   DD  DSN=&&FONTLIB,DISP=SHR,                          *
*  //             UNIT=SYSDA,VOL=REF=*.INLNFONT                    *
*  //         DD  DSN=AFPDEMO.PROD.FONTLIB,DISP=SHR                *
*  //         DD  DSN=SYS1.FONT3820,DISP=SHR                       *
*                                                                  *
*  This exit receives control from PSF at FSA initialization and   *
*  sets up the environment required by APSUX04 to divert inline    *
*  overlays, page segments, coded fonts, character sets, and code  *
*  pages to the temporary libraries.  Subsequently, this exit      *
*  receives control at the beginning of each print dataset in order *
*  to delete any objects written to the temporary datasets by the  *
*  preceding job.  Control is also received at FSA termination to  *
*  close files and release GETMAINed storage.                      *
*                                                                  *
*  APSUX04 is responsible for detecting the presence of an inline  *
*  resource in the datastream and writing it to the appropriate    *
*  dataset.   This module is responsible for the following:        *
*                                                                  *
*  On PSF INITIALIZATION:                                          *
*                                                                  *
*    1. GETMAIN and initialize control areas and write buffers.    *
*    2. Open files.                                                *
*                                                                  *
*  At Beginning of Each New Dataset:                               *
*                                                                  *
*    1. Check to see if objects have been written into any of      *
*       the temporary libraries by the previous job.               *
*    2. If objects were written by the previous job, use STOW      *
*       to delete these members from the library.                  *
*                                                                  *
*  At FSA Termination:                                             *
*                                                                  *
*    1. Check to see if objects have been written into any of      *
*       the temporary libraries by the previous job and delete     *
*       them if there were.                                        *
*    2. Close files and FREEMAIN storage.                          *
*                                                                  *
*                                                                  *
*  This exit also requests control at access time for pagedefs and *
*  formdefs so that it can force a reload from dasd.  This function*
*  is completely independent of the inline resource handling       *
*  function.                                                       *
*                                                                  *
*  The basic logic and environment for PSF exits are described in  *
*  the PSF System Programming Guide (S544-3672).                   *
*                                                                  *
*                                                                  *
*               ########################################           *
*               #                                      #           *
*               #  WARNING * WARNING * WARNING * WARNING  #        *
*               #               RESTRICTIONS           #           *
*               #                                      #           *
*               ########################################           *
*                                                                  *
*  This exit has the following restrictions:                       *
*                                                                  *
```

```
*                                                                     *
*     -  While this exit has been coded to be re-entrant, it should   *
*        not be employed in a PSF address space that is supporting    *
*        multiple FSAs (i.e. printers).  This sample coding employs   *
*        a single set of temporary resource libraries.  If more than  *
*        one printer is active in the address space, then exit        *
*        processing done on behalf of one printer may delete or       *
*        overlay objects required for another.  The logic could be    *
*        adapted to a multiple printer situation by adding            *
*        additional temporary file DCBs, one set of 3 for each        *
*        printer driven by the address space.  They would have to     *
*        have different DCB names.  Also, the control block areas     *
*        that pass DCB addresses back and forth would have to be      *
*        updated to reflect any DCBs added to the exit coding.        *
*                                                                     *
*     -  There is another reason to avoid driving multiple printers   *
*        per address space when employing this coding.  Normal PDS    *
*        facilities (i.e. BLDL and STOW) are used to add and delete   *
*        members.  While, at any given moment, the temporary          *
*        datasets contain only members provided in the datastream of  *
*        the current dataset, over time a printer supporting a high   *
*        level of inline resource activity may cause these datasets   *
*        to expand beyond 16 extents or ask for additional space      *
*        that is not available (i.e. incur an x37 ABEND).  If a       *
*        single printer is supported in the address space, and the    *
*        JCL coding shown above is employed (i.e. using temporary     *
*        datasets) then this situation can be fixed by simply         *
*        cancelling the PSF address space and restarting the          *
*        printer.  For MVS/ESA systems operating at the required      *
*        level of DFP, the new PDSE support could be used to          *
*        circumvent this problem completely.                          *
*                                                                     *
*     -  The exit logic tacitly assumes that all resources will be    *
*        deleted by PSF at the end of each print dataset.  This is    *
*        not true in the case of fonts.  The default algorithms used  *
*        by PSF to manage fonts permit them to remain in the printer  *
*        across dataset boundaries.  Thus, if an inline font named    *
*        GT10 is loaded into the printer, it may not be deleted at    *
*        the end of the dataset and may become accessible to a        *
*        subsequent job.  If this is undesirable, code could be added*
*        to APSUX07 to force the deletion of all resources loaded     *
*        from the temporary libraries.                                *
*                                                                     *
*     -  The converse of the above may also be a problem.  For        *
*        instance, if a font GT10 is included inline, but another     *
*        font GT10 already exists in the printer, then PSF will not   *
*        load the "inline" font.  This is because PSF does not        *
*        perceive the font as being inline, but rather as coming      *
*        from its normal library.  Again, this problem should be      *
*        addressable with additional code.  APSUX07 can set the       *
*        RLSTLOAD flag to cause a reload from DASD.                   *
*                                                                     *
*     -  In repositioning situations, the Begin Data Set call of      *
*        APSUX07 is not always executed.  Imagine the case of two     *
*        print datasets, A and B.  PSF sends the final pages of       *
*        dataset A to the printer which absorbs them into its page    *
*        buffer and starts to physically print them.  While the       *
*        printer works on these pages, PSF starts working on dataset  *
*        B, which contains some inline resources.  These resources    *
*        are diverted into the temporary datasets by APSUX04.  But    *
*        before all of the last pages of dataset A can be physically  *
*        printed, a printer error occurs requiring PSF to reposition  *
*        backwards beyond the start of dataset B into the last pages  *
*        of dataset A.  Now, however, if dataset A references a       *
*        resource with the same name as one loaded by dataset B into  *
*        the temporary libraries, the "wrong" copy may be loaded.     *
```

```
*                                                                     *
*        This is an unlikely situation, but it is possible.  Again,   *
*        additional code should provide a resolution.  For instance,  *
*        one possibility might be to change the name of diverted      *
*        objects so as to make them inaccessible without action by    *
*        APSUX07.  Then APSUX07 could, in turn, ensure that only the  *
*        owning print dataset got access to the renamed resource.     *
*                                                                     *
*     -  This exit uses a number of MVS system service macros.  The   *
*        coding as written does not do error checking or result       *
*        validation on return from these calls.  Therefore, an IO     *
*        error or other recoverable failure could result in an        *
*        ABEND of the PSF address space.                              *
*                                                                     *
*     -  This exit will ONLY run AMODE 24, RMODE 24.                  *
*                                                                     *
*     -  If PSF is not marked non-swappable in the PPT, unpredictable*
*        abends may occur.                                            *
*                                                                     *
*                                                                     *
***********************************************************************
APSUX07  START 0
         TITLE 'DSECT - GEXTP'
         APSGEXTP LIST=YES
         TITLE 'DSECT - UECA'
         APSUECA  LIST=YES
         TITLE 'DSECT - RLST'
         APSURLST LIST=YES
         TITLE 'APSUX07 - RESOURCE EXIT:  INLINE RESOURCE HANDLING'
APSUX07  CSECT ,
APSUX07  AMODE 24
APSUX07  RMODE 24
         USING *,R15
         B     START
         DC    AL1(16)                 LENGTH OF FOLLOWING FIELDS
         DC    CL8'APSUX07 '           NAME OF THIS ROUTINE
         DC    CL8'&SYSDATE'           DATE OF THIS ASSEMBLY
         DC    CL16'SOURCEM: V2X7ILR'
         DROP  R15
START    DS    0H
         STM   R14,R12,R12(R13)        SAVE CALLERS REGISTERS
         USING APSUX07,BASEREG         NEW ADDRESSABILITY
         LR    BASEREG,R15             SWITCH BASE REGISTER
*
***********************************************************************
* Get basing for control blocks                                       *
***********************************************************************
         USING APSGEXTP,GEXTPTR        SET ADDRESSABILITY TO GEXTP
         USING XTP7,XTP7PTR            SET ADDRESSABILITY TO XTP7
         USING APSUECA,ECAPTR          SET ADDRESSABILITY TO APSUECA
         USING APSURLST,RLSTPTR        SET ADDRESSABILITY TO APSURLST
         L     GEXTPTR,0(,R1)          LOAD ADDRESS OF GEXTP
         L     ECAPTR,XTPECAP          LOAD ADDRESS OF APSUECA
         LR    R2,R13                  LOAD ADDRESS OF CALLERS SAVE
         LA    R13,ECAUSAVE            ADDRESS OF APSUX04 SAVE AREA
         ST    R2,4(,R13)              SAVE CALLERS SAVE AREA ADDRESS
         ST    R13,8(,R2)              SAVE APSUX04 SAVE AREA ADDRESS
         L     XTP7PTR,XTPRECP         LOAD ADDRESS OF XTP7
         L     RLSTPTR,XTP7LSTP        LOAD ADDRESS OF APSURLST
*
***********************************************************************
*  Determine what type of call is being made to the exit.  If not     *
*  an initialization call or an access call, establish addressability*
*  to the GETMAINed work areas for the other routines.                *
***********************************************************************
CALLTYPE EQU   *
```

```
         TM    XTP7ETYP,INITCALL       INITIALIZATION CALL?
         BO    INITRTN                  YES, GOTO INIT ROUTINE
         TM    XTP7ETYP,ACCCALL        RESOURCE ACCESS TIME CALL?
         BO    ACCRTN                   YES, GOTO ACCESS ROUTINE
         L     ILRPTR,ECALEN           ILRPTR = LEN OF ENTIRE ECA       00107516
         LA    R3,ILRCOMML             R3 = LEN OF ILRCOMM AREA         00107616
         SLR   ILRPTR,R3               ILRPTR = ECALEN-LEN OF ILRCOMM   00107716
         LA    ILRPTR,APSUECA(ILRPTR)  ILRPTR -> ILRCOMM AREA           00107816
         L     ILRPTR,0(ILRPTR)        ILRPTR -> GETMAINED WORK AREA
         USING ILRWORK,ILRPTR          ADDRESSABILITY TO ILRWORK AREA
         TM    XTP7ETYP,BDSCALL        BEGIN DATASET CALL?
         BO    BDSRTN                   YES, GOTO BDS ROUTINE
         TM    XTP7ETYP,TRMCALL        TERMINATION OF FSA CALL?
         BO    TERMRTN                  YES, GOTO TERM ROUTINE
         B     GETOUT                  THIS SHOULD NEVER HAPPEN
*
*********************************************************************
*  INITIALIZATION ROUTINE:                                          *
*    FUNCTION:                                                      *
*      - Request control at access time for formdefs and pagedefs   *
*      - Initialize communication area for APSUX04                  *
*      - Getmain a workarea and copy DCBs, OPEN parm lists and      *
*        other control blocks needed by APXUX04 into it             *
*      - Open files                                                 *
*      - Initialize BLDL list area                                  *
*********************************************************************
INITRTN  EQU   *
*SWTL-----------------TRACING WTL-----------------------------------*
*        USING WTLDSECT,R1
*        MVC   ECAWKBUF+100(WTLLISTL),WTLLIST  MOVE WTLLIST TO WORK
*        LA    R1,ECAWKBUF+100         R1 -> WTL LIST FORM
*        MVC   WTLTEXT,=CL47'INITIALIZING INLINE RESOURCE EXITS FOR'
*        MVC   WTLTEXT+39(8),XTP7PNAM  PUT PRINTER NAME IN WTL MSG
*        WTL   MF=(E,(1))
*EWTL-----------------END TRACING WTL-------------------------------*
         MVI   XTP7NACC,XTP7AFD+XTP7APD REQUEST CONTROL AT ACCESS
*                                      TIME FOR FDEFS AND PDEFS
         MVI   XTP7MISC,XTP7ETRM+XTP7EBDS AND ALSO AT FSA TERMINATION
*                                       AND BEGINNING OF DATASET
*
*-------------------------------------------------------------------*
*  GETMAIN an area for control blocks and place its address in the  *
*  last word of the ECAWKBUF area.  This pointer will be visible to *
*  APSUX04.                                                         *
*-------------------------------------------------------------------*
         L     ILRPTR,ECALEN           ILRPTR = LEN OF ENTIRE ECA       00107516
         LA    R3,ILRCOMML             R3 = LEN OF ILRCOMM AREA         00107616
         SLR   ILRPTR,R3               ILRPTR = ECALEN-LEN OF ILRCOMM   00107716
         LA    ILRPTR,APSUECA(ILRPTR)  ILRPTR -> ILRCOMM AREA           00107816
         USING ILRCOMM,ILRPTR          ADDRESSABILITY TO ILRCOMM AREA
         GETMAIN RU,LV=4000            GET CONTROL BLOCK WORK AREA
         ST    R1,ILRCOMM              SAVE POINTER FOR UX04
*
*-------------------------------------------------------------------*
*  Now initialize GETMAINed area for use by UX04.  Move in IO control *
*  blocks, open temporary object files, GETMAIN a write buffer area,  *
*  and initialize pointers to a BLDL list for use by UX04.          *
*-------------------------------------------------------------------*
         LR    ILRPTR,R1               ILRPTR -> GETMAINED AREA
         USING ILRWORK,ILRPTR
         LA    R2,ILRWORK              R2 -> MOVE TARGET
         LA    R3,ILRBLEN              R3 = MOVE LENGTH
         LA    R4,ILRLITS              R4 -> MOVE SOURCE
         LR    R5,R3                   R5 = MOVE LENGTH
         MVCL  R2,R4                   MOVE CTL BLKS TO WORK AREA
*-------------------------------------------------------------------*
```

```
*  The control blocks we just moved into the GETMAINed area don't have*
*  the DCB addresses, etc. needed by UX04, we have to store them now. *
*--------------------------------------------------------------------*
         LA    R4,OVLYDCB              R4 -> DCB FOR TEMP OVERLAYS
         STCM  R4,7,OVLDCBAD+1         SAVE ADDR FOR UX04
         STCM  R4,7,OPENLST+1            AND IN OPEN PARM LIST
         LA    R4,PSEGDCB              R4 -> DCB FOR TEMP PAGE SEGMENTS
         STCM  R4,7,PSGDCBAD+1         SAVE ADDR FOR UX04
         STCM  R4,7,OPENLST+5            AND IN OPEN PARM LIST
         LA    R4,FONTDCB              R4 -> DCB FOR TEMP FONTS
         STCM  R4,7,FCSDCBAD+1         SAVE ADDR FOR UX04
         STCM  R4,7,FCPDCBAD+1         SAVE ADDR FOR UX04
         STCM  R4,7,FCFDCBAD+1         SAVE ADDR FOR UX04
         STCM  R4,7,OPENLST+9            AND IN OPEN PARM LIST
         LA    R1,OPENLST              R1 -> OPEN PARM LIST
         OPEN  ,MF=(E,(1))             OPEN 3 TEMP LIBRARIES
         GETMAIN RU,LV=WRTBUFLN        GET OUTPUT BUFFER
         ST    R1,WRTBUFAD             SAVE POINTER FOR UX04
         LR    R2,R1                   R2 -> WRITE BUFFER AREA
         L     R3,WRTBUFLW             R3 = LENGTH OF GETMAINED AREA
         BCTR  R3,0                    DECR BY 1 JUST TO BE SAFE
         SLR   R5,R5                   CLEAR R5 TO ZEROS
         MVCL  R2,R4                   CLEAR WRITE BUFFER TO ZEROS
         LA    R4,BLDLNT1-BLDLELEN     R4 -> 1ST BLDL ENTRY -1 ENTRY
         ST    R4,BLDLLAST             SAVE POINTER (IND 0 ENTRIES)
         LA    R4,WRITELST             R4 -> WRITE PARM LIST
         ST    R4,WRTLSTAD             SAVE POINTER FOR UX04
         B     GETOUT                  EXIT APSUX07
*
**********************************************************************
* ACCESS ROUTINE:                                                    *
*   (NOTE: This routine is not related to the inline resource        *
*          handling functions.)                                      *
*   FUNCTION:                                                        *
*     - Check dataset type and don't process headers, trailers, msgds*
*     - Set RLSTLOAD flag to force pagedef/formdef reload from DASD  *
*       for non-inline resources.  PSF won't use virtual storage     *
*       copy.                                                        *
**********************************************************************
ACCRTN   EQU   *
         TM    XTP7DSAT,XTP7PDFT+XTP7PJHD+XTP7PJTR+XTP7PDSH+XTP7PMDS
*                   TEST FLAGS FOR NON-USER DATASET
         BNZ   GETOUT                  EXIT IF NON-USER
         OI    RLSTAFLG,RLSTLOAD       REQUEST RESOURCE LOAD FROM DASD
*                                       FOR ALL PAGEDEFS AND FORMDEFS
         B     GETOUT                  SCRAM, WE'RE DONE
*
**********************************************************************
* BDS ROUTINE:                                                       *
*   FUNCTION:                                                        *
*     - Check to see if previous job has stored inline resources     *
*       in the temp libraries, if so we delete them using STOW.      *
*       If no resources exist in any temporary library, then the     *
*       address in the BLDLLAST field will be < address of first     *
*       available BLDL entry.                                        *
**********************************************************************
BDSRTN   EQU   *
         MVI   ILRFLAGS,0              CLEAR ILRFLAGS FIELD
         TM    XTP7DSAT,XTP7PDFT+XTP7PJHD+XTP7PJTR+XTP7PDSH+XTP7PMDS
*                   TEST FLAGS FOR NON-USER DATASET
         BNZ   GETOUT                  EXIT IF NON-USER
         LA    R4,BLDLNT1              R4 -> FIRST BLDL ENTRY
         L     R3,BLDLLAST             R3 -> LAST BLDL ENTRY
         CR    R3,R4                   BLDLLAST -> B4 START OF LIST?
         BL    GETOUT                  IF YES, EXIT - NO BLDL ENTRIES
         L     R2,BLDLLEN              R3 = LENGTH OF EACH ENTRY
```

```
         USING BLDLNTRY,R4             ADDRESSABILITY FOR BLDL DSECT
DELMBR   EQU   *
         L     R1,BLDLDCB              R1 -> DCB FOR MEMBER DELETE
         STOW  (1),(4),D               DELETE MEMBER
         BXLE  R4,R2,DELMBR            LOOP TO NEXT MEMBER
         LA    R2,BLDLNT1-BLDLELEN     R2 -> START OF LIST -1 ENTRY
         ST    R2,BLDLLAST             RESET TO INDICATE NO ENTRIES
         B     GETOUT                  SCRAM, WE'RE DONE
         DROP  R4
*********************************************************************
*  FSA TERMINATION ROUTINE:                                         *
*    FUNCTION:                                                      *
*      - Delete any members in temp libraries                       *
*      - Close files and free getmained areas                       *
*********************************************************************
TERMRTN  EQU   *
         LA    R4,BLDLNT1              R4 -> FIRST BLDL ENTRY
         L     R3,BLDLLAST             R3 -> LAST BLDL ENTRY
         CR    R3,R4                   BLDLLAST -> B4 START OF LIST?
         BL    CLOSEM                  IF YES, EXIT - NO BLDL ENTRIES
         L     R2,BLDLLEN              R3 = LENGTH OF EACH ENTRY
         USING BLDLNTRY,R4             ADDRESSABILITY FOR BLDL DSECT
DELMBR1  EQU   *
         L     R1,BLDLDCB              R1 -> DCB FOR MEMBER DELETE
         STOW  (1),(4),D               DELETE MEMBER
         BXLE  R4,R2,DELMBR1           LOOP TO NEXT MEMBER
         LA    R2,BLDLNT1-BLDLELEN     R2 -> START OF LIST -1 ENTRY
         ST    R2,BLDLLAST             RESET TO INDICATE NO ENTRIES
         DROP  R4
CLOSEM   EQU   *
         LA    R1,OPENLST              R1 -> OPEN PARM LIST
         CLOSE ,MF=(E,(1))             CLOSE ALL FILES
         L     R1,WRTBUFAD             R1 -> WRITE BUFFER AREA
         FREEMAIN RU,LV=32700,A=(1)    FREE BUFFER
         FREEMAIN RU,LV=4000,A=(ILRPTR)  FREE WORKAREA
         B     GETOUT                  TIME TO LEAVE
*
*********************************************************************
*  EXIT LINKAGE                                                     *
*********************************************************************
GETOUT   EQU   *
         SLR   R15,R15                 PSF EXPECTS ZERO RETURN CODE
         L     R13,4(,R13)             RESTORE CALLERS SAVE AREA ADDR.
         L     R14,12(,R13)            RESTORE CALLERS RETURN ADDRESS
         LM    R0,R12,20(R13)          RESTORE CALLERS REGISTERS
         BR    R14                     RETURN TO CALLER
*
*********************************************************************
*  EQUATES TO VERIFY TYPE OF EXIT CALL, COMPARE WITH XTP7ETYP       *
*********************************************************************
INITCALL EQU   B'10000000'     RESOURCE MANAGER INITIAL CALL
BDSCALL  EQU   B'01000000'     BEGINNING OF DATA SET CALL
ACCCALL  EQU   B'00100000'     RESOURCE ACCESS
LDBCALL  EQU   B'00010000'     RESOURCE LOAD BEGIN
LDECALL  EQU   B'00001000'     RESOURCE LOAD END
DSECALL  EQU   B'00000100'     RESOURCE DELETE AT DATA SET END
TRMCALL  EQU   B'00000010'     TERMINATION OF FSA
*********************************************************************
* EQUATES TO VERIFY TYPE OF RESOURCE CALL, COMPARE WITH XTP7RTYP    *
*********************************************************************
PDEF     EQU   B'10000000'              PAGEDEF
FDEF     EQU   B'01000000'              FORMDEF
FONT     EQU   B'00100000'              CODED FONT
OVLY     EQU   B'00010000'              MEDIUM OVERLAY
PSEG     EQU   B'00001000'              PAGE SEGMENT
*********************************************************************
```

```
* EQUATES FOR RLSTATTR FLAG FIELD                                      *
************************************************************************
DFLT     EQU   B'10000000'              DEFAULT FD/PD RESOURCE
MDFLT    EQU   B'01000000'              MODIFIED DEFAULT FD/PD RESOURCE
INLINE   EQU   B'00100000'              INLINE FD/PD RESOURCE
SOFTPS   EQU   B'00010000'              SOFT PAGE SEGMENT
MEFONT   EQU   B'00001000'              MULTIPLE ENTRY FONT
************************************************************************
*        REGISTER EQUATES                                              *
************************************************************************
R0       EQU   0
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9
R10      EQU   10
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
GEXTPTR  EQU   R7
XTP7PTR  EQU   R8
ECAPTR   EQU   R9
RLSTPTR  EQU   R10
ILRPTR   EQU   R11
BASEREG  EQU   R12
*                                                                      *
************************************************************************
*  WRITE BUFFER LENGTH, FOR GETMAIN AND CLEARING CODE
************************************************************************
*
WRTBUFLN EQU   37000
WRTBUFLW DC    AL4(WRTBUFLN)
*                                                                      *
************************************************************************
*  RELOCATABLE CONTROL BLOCKS FOR IO, BLDL, ETC.
*      The following DCs define the constant data that will be
*      moved into the GETMAINed work area by the INITRTN.  The first
*      11 words are pointers that will be initialized with the proper
*      addresses after the control blocks have been moved into the the
*      work area.  Following these pointers are the DCBs, the WRITE
*      list-form macro, and the first BLDL entry in the work area.
*      Once moved, this data is referenced by ILRPTR and mapped by
*      DSECT ILRWORK.
************************************************************************
*
ILRLITS  DS    0D
ILRL0    DC    AL4(0)                  RESERVED FOR UX04 FLAGS
ILRL1    DC    AL4(0)                  POINTER TO WRITE BUFFER
ILRL2    DC    AL4(0)                  POINTER TO OPEN PARM LIST
*------------------Start Of DCB Pointer Table --------------------*
*       The following five words are used as a search table by UX04.
*       The high-order byte of each word contains the code that is
*       used in a Begin Resource record to identify the type of
*       resource that is to be written to the DCB having its address
*       in the remaining 3 bytes of the word.  There are 3 types
*       of font resource and they are all written to the same
*       physical file.  However, there is a separate DCB pointer for
*       each type of font object (all pointing to the same DCB) to
*       make the search logic in UX04 more generic.
```

```
*-------------------------------------------------------------------*
ILRL3A   DC    AL1(BROVLY)             FLAG FOR INLINE OVERLAY
ILRL3B   DC    AL3(0)                  POINTER TO OVERLAY DCB
ILRL4A   DC    AL1(BRPSEG)             FLAG FOR INLINE PSEG
ILRL4B   DC    AL3(0)                  POINTER TO PSEG DCB
ILRL5A   DC    AL1(BRCSET)             FLAG FOR INLINE CHARACTER SET
ILRL5B   DC    AL3(0)                  POINTER TO FONT DCB
ILRL6A   DC    AL1(BRCPAGE)            FLAG FOR INLINE CODE PAGE
ILRL6B   DC    AL3(0)                  POINTER TO FONT DCB
ILRL7A   DC    AL1(BRCFONT)            FLAG FOR INLINE CODED FONT
ILRL7B   DC    AL3(0)                  POINTER TO FONT DCB
*------------------- End Of DCB Pointer Table ---------------------*
ILRL8    DC    AL4(0)                  POINTER TO WRITE PARM LIST
ILRL9    DC    AL4(0)                  POINTER TO LAST BLDL ENTRY
ILRL10   DC    AL4(BLDLELEN)           LENGTH OF EACH BLDL ENTRY
OPENLIT  OPEN  (,(OUTPUT),,(OUTPUT),,(OUTPUT)),MF=L
         DS    0F
OPENLITL EQU   *-OPENLIT
OVLYLIT  DCB   MACRF=W,DSORG=PO,RECFM=VBM,DDNAME=INLNOVLY,             X
               LRECL=8205,BLKSIZE=8209
         DS    0F
OVLYLN   EQU   *-OVLYLIT
PSEGLIT  DCB   MACRF=W,DSORG=PO,RECFM=VBM,DDNAME=INLNPSEG,             X
               LRECL=8205,BLKSIZE=8209
         DS    0F
PSEGLN   EQU   *-PSEGLIT
FONTLIT  DCB   MACRF=W,DSORG=PO,RECFM=VBM,DDNAME=INLNFONT,             X
               LRECL=8205,BLKSIZE=8209
         DS    0F
FONTLN   EQU   *-FONTLIT
WRITELIT WRITE OUTDECB,SF,MF=L
         DS    0F
WRITELN  EQU   *-WRITELIT
BLDLLITE EQU   *
         DC    CL8' '          BLANK NAME ENTRY
         DC    XL4'00000000'   BLDL FLAGS
         DC    XL4'00000000'   DCB ADDRESS FOR THIS MEMBER
BLDLELEN EQU   *-BLDLLITE
ILRBLEN  EQU   *-ILRLITS
         DC    F'0'
**********************************************************************
WTLLIST  WTL   'APSUX07:                                              ',*
               MF=L
WTLLISTL EQU   *-WTLLIST
**********************************************************************
*         BEGIN RESOURCE RECORD FLAGS FOR RESOURCE TYPE
**********************************************************************
BRCSET   EQU   X'40'     CHARACTER SET FLAG
BRCPAGE  EQU   X'41'     CODE PAGE FLAG
BRCFONT  EQU   X'42'     CODED FONT FLAG
BRPSEG   EQU   X'FB'     PAGE SEGMENT FLAG
BROVLY   EQU   X'FC'     OVERLAY FLAG
BRPDEF   EQU   X'FD'     PAGEDEF FLAG
BRFDEF   EQU   X'FE'     FORMDEF FLAG
*                                                                     *
**********************************************************************
* DSECT FOR ILR COMM AREA AT END OF UECA ECAWKBUF FIELD              *
**********************************************************************
ILRCOMM  DSECT
ILRPARMS DS    F          POINTER TO GETMAINED PARAMETER AREA
ILRCOMML EQU   *-ILRCOMM  LENGTH OF ILR COMM AREA
*
**********************************************************************
*          ILRWORK  CONTROL BLOCKS DSECT STARTS HERE
*             THIS MAPS THE GETMAINED WORK AREA
**********************************************************************
```

```
*
ILRWORK  DSECT
ILRFLAGS DS    AL4                APSUXO4 FLAG AREA
WRTBUFAD DS    AL4                POINTER TO OUTPUT BUFFER
OPENLAD  DS    AL4                POINTER TO OPEN PARM LIST
OVLDCBAD DS    AL4                POINTER TO OVERLAY DCB
PSGDCBAD DS    AL4                POINTER TO PSEG DCB
FCSDCBAD DS    AL4                POINTER TO FONT DCB (CHAR SETS)
FCPDCBAD DS    AL4                POINTER TO FONT DCB (CODE PAGES)
FCFDCBAD DS    AL4                POINTER TO FONT DCB (CODED FONTS)
WRTLSTAD DS    AL4                POINTER TO WRITE PARM LIST
BLDLLAST DS    AL4                POINTER TO LAST BLDL ENTRY
BLDLLEN  DS    AL4                LENGTH OF EACH BLDL ENTRY
OPENLST  EQU   *                      OPEN PARAMETER LIST
         ORG   OPENLST+OPENLITL
OVLYDCB  DS    OF                     DCB FOR INLINE OVERLAY FILE
         ORG   OVLYDCB+OVLYLN
PSEGDCB  DS    OF                     DCB FOR INLINE PSEG FILE
         ORG   PSEGDCB+PSEGLN
FONTDCB  DS    OF                     DCB FOR INLINE FONT FILE
         ORG   FONTDCB+FONTLN
WRITELST EQU   *
         ORG   WRITELST+WRITELN
BLDLLIST EQU   *
BLDLNT1  DS    CL12       BLDL LIST ENTRY
         DS    CL4        DCB FOR FILE MEMBER WAS WRITTEN TO
*********************************************************************
*          BLDL ENTRY DSECT STARTS HERE
*               This DSECT is used when deleting members from
*               the temporary datasets with STOW.  The member name
*               and DCB address have been filled in by UX04 when the
*               member was written into the dataset.
*********************************************************************
*
BLDLNTRY DSECT              DSECT FOR ENTRIES
BLDLENT  DS    OCL16      BLDL LIST ENTRY
BLDLMNAM DS    CL8        MEMBER NAME
BLDLFLGS DS    XL4        FLAG WORD
BLDLDCB  DS    XL4        ADDRESS OF DCB OF FILE CONTAINING MEMBER
*                                                                   *
*********************************************************************
*      DSECT TO MAP LIST FORM WTL BUILT WITH WTL MF=(L)             *
*********************************************************************
WTLDSECT DSECT
         DS    F
         DS    CL9
WTLTEXT  DS    CL47
         END
```

A.5 ILRPACK Program

This utility program may be used to create a print file with an inline resource group. See A.3, “PSF/MVS Inline Resource Exit APSUX04” on page 77 for a discussion on inline resource groups.

A.5.1 Sample Assembler Code

```
         TITLE 'AFP INLINE RESOURCE PACKER'
ILRPACK  CSECT
*********************************************************************
*   This program reads a control file containing a list of names of  *
*   AFP resource objects.  The resource objects are retrieved from   *
*   the appropriate library and written to the output print file     *
*   (ILROUT) as an AFP resource group.    Then the input print file  *
*   (ILRIN) is copied to the print output file after the resource    *
*   group.  When this file is shipped to a PSF/VM (or a PSF/MVS with *
*   the inline resource exits installed) the resources packed inline *
*   by this program will be used in preference to accessing resources*
*   in the normal resource libraries defined to that PSF.  Therefore,*
*   the file becomes "portable" from system to system.               *
*                                                                   *
*   DEFAULT FILE DDS:                                               *
*   -----------------                                               *
*      The DDNAMES shown below are the defaults.  The ILRSYSIN ddname*
*   may be changed with an EXEC parm, like so:                      *
*                                                                   *
*            //STEP1  EXEC PGM=ILRPACK,PARM='ILRSYSIN=NEWNAME'      *
*                                                                   *
*   which would cause the file with ddname NEWNAME to be opened as  *
*   the input control file.  The other ddnames may be changed by    *
*   entering a command in the ILRSYSIN (or whatever) input file.    *
*   Here are the default ddnames:                                   *
*                                                                   *
*      //ILRSYSIN DD  POINTS TO INPUT CONTROL FILE                 *
*      //ILRIN    DD  POINTS TO INPUT PRINT FILE                   *
*      //ILROUT   DD  POINTS TO OUTPUT PRINT FILE                  *
*      //ILRSYSPR DD  POINTS TO OUTPUT REPORT FILE                 *
*      //ILRPDEF  DD  POINTS TO INPUT PAGEDEF LIBRARY              *
*      //ILRFDEF  DD  POINTS TO INPUT FORMDEF LIBRARY              *
*      //ILROVLY  DD  POINTS TO INPUT OVERLAY LIBRARY              *
*      //ILRPSEG  DD  POINTS TO INPUT PAGESEG LIBRARY              *
*      //ILRFONT  DD  POINTS TO INPUT FONT    LIBRARY              *
*                                                                   *
*   If resources of a particular type (e.g. fonts or overlays) are  *
*   not required, the ddname defining that library may be omitted.  *
*   Do not allocate unnecessary libraries to 'DD DUMMY', an ABEND   *
*   will result.                                                    *
*                                                                   *
*   INPUT CONTROL FILE SYNTAX:                                      *
*   --------------------------                                      *
*      User requests to pack objects into an Inline Resource Group  *
*   are organized into "request groups".  Each request group defines *
*   the input file, the output file, and the resource objects that   *
*   are to be packed into the output file with the original input    *
*   file.  At the beginning of each request group, all the input and *
*   output files with the exception of ILRSYSIN itself may be       *
*   redefined.  If no ddname redefinitions are encountered the       *
*   defaults above are used.  For all request groups but the first,  *
*   it is mandatory that, at a minimum, new input and output files   *
*   be defined.  All other ddnames that are not redefined will       *
*   remain as they were.                                            *
*                                                                   *
*      Two types of command may be entered. The first type specifies *
*   the ddname to use for any of the program files except for       *
*   ILRSYSIN. The second type identifies a resource type and the     *
*   name of an object of that type to be retrieved from the         *
*   currently active library and packed into a resource group at the *
*   beginning of the print output dataset.                          *
*                                                                   *
*      All ddname definitions must precede the list of object names  *
*   to be retrieved using those DD definitions.  Once object        *
*   retrieval has started, encountering a ddname definition causes   *
*   the current resource group to be ended, the current input file   *
```

```
*   to be copied to output, and all files except ILRSYSIN to be      *
*   closed. Then the new dd definition cards are read, new ddnames   *
*   assigned, and the files reopened.  This is assumed to represent  *
*   a request for a new output file.  Therefore at a minimum, the    *
*   ddnames for ILRIN and ILROUT must be redefined.  Otherwise, the  *
*   reopen of these DCBs will fail and the program terminates with a *
*   return code of 20.  All other ddnames may remain the same if     *
*   desired.                                                         *
*                                                                    *
*   Below is the syntax of the control cards.  Note that fields are  *
*   positional and must appear in the columns indicated.             *
*                                                                    *
*     DD DEFINITION SYNTAX:                                          *
*       COL1     COL9                                                *
*       |        |                                                  *
*       V        V                                                  *
*       DDIDENT  =NEW DDNAME                                        *
*                                                                    *
*          Where DDIDENT is one of:                                  *
*                ILRPDEF   IDENTIFIES PAGE DEFINITION LIBRARY        *
*                ILRFDEF   IDENTIFIES FORM DEFINITION LIBRARY        *
*                ILROVLY   IDENTIFIES OVERLAY LIBRARY                *
*                ILRPSEG   IDENTIFIES PAGE SEGMENT LIBRARY           *
*                ILRFONT   IDENTIFIES FONT LIBRARY                   *
*                ILRIN     IDENTIFIES INPUT PRINT FILE               *
*                ILROUT    IDENTIFIES OUTPUT PRINT FILE              *
*                ILRSYSPR  IDENTIFIES REPORT LISTING FILE            *
*                                                                    *
*          Where new ddname is the ddname the program is to use for  *
*                for the current request                             *
*                                                                    *
*                                                                    *
*     RESOURCE PACKING REQUEST SYNTAX:                               *
*       COL 1     COL 10       COL 19                                *
*       |         |            |                                     *
*       V         V            V                                     *
*       OBJTYPE   OBJECT NAME  NEST=OV,PS,FO                         *
*                                                                    *
*          Where OBJTYPE = ONE OF:  PAGEDEF = PAGE DEFINITION        *
*                                   FORMDEF = FORM DEFINITION        *
*                                   OVERLAY = OVERLAY                *
*                                   PAGESEG = PAGE SEGMENT           *
*                                   CFONT   = CODED FONT             *
*                                   CPAGE   = CODE PAGE              *
*                                   CHARSET = CHARACTER SET          *
*                                   SCANFILE= SCAN THE INPUT FILE FOR*
*                                             RESOURCE REFERENCES    *
*                                                                    *
*          Where OBJECT NAME = member name of object (including      *
*                              prefix) in resource library           *
*                                                                    *
*          Where NEST= is an optional parameter that indicates that  *
*                      "nested" resources are to be retrieved.       *
*                      When one resource object internally           *
*                      references another, that is termed a          *
*                      nested reference.  When this parameter is     *
*                      coded, objects of the type(s) listed in       *
*                      the NEST= parameter that are referenced       *
*                      from within the object named in the object    *
*                      name field will be retrieved and packed. If   *
*                      the objtype field is "scanfile", then this    *
*                      parameter must be coded.  Legal values        *
*                      are:                                          *
*                         NEST=OV   pack referenced OVERLAYS         *
*                         NEST=PS   pack referenced PAGE SEGMENTS    *
*                         NEST=FO   pack referenced FONTS (implies   *
```

```
*                                  all 3 types of font object; code *
*                                  pages, character sets, and coded *
*                                  fonts). This causes both         *
*                                  bounded and unbounded (i.e. 3820 *
*                                  and 3800) versions of referenced *
*                                  fonts to be retrieved if present *
*                                  in the font library).            *
*                         NEST=BB  pack referenced BOUNDED BOX      *
*                                  FONTS (same as nest=fo except    *
*                                  only 3820 fonts are retrieved)   *
*                         NEST=UB  pack referenced UNBOUNDED BOX    *
*                                  FONTS (same as nest=fo except    *
*                                  only 3800 fonts are retrieved)   *
*                                                                   *
*                                                                   *
*     A sample ILRSYSIN control file might look like the following, *
*     the numbers to the left of the statements correspond to the   *
*     following notes that describe what that statement does:       *
*       //ILRSYSIN DD *                                           *
*   1.  ILRSYSPR=XXXPRINT                                         *
*   2.  ILROUT  =MYOUTPUT                                         *
*   3.  PAGEDEF  P1TINLN1 NEST=FO,PS                              *
*   4.  FORMDEF  F1TINLN1 NEST=OV,PS,FO                           *
*   5.  OVERLAY  O1TINLN1 NEST=PS                                 *
*   6.  PAGESEG  S1TINLN1                                         *
*   7.  ILRIN   =INPUT2                                           *
*   8.  ILROUT  =OUTPUT2                                          *
*   9.  SCANFILE          NEST=PS,BB                              *
*  10.  PAGEDEF  P1JUNK   NEST=PS,BB                              *
*  11.  FORMDEF  F1TRASH                                         *
*  12.  ILRIN   =AFPDSIN                                         *
*  13.  ILROUT  =OUTPUT3                                         *
*  14.  SCANFILE          NEST=PS                                 *
*       /*                                                       *
*                                                                   *
*   1.  Use ddname XXXPRINT for the output report                 *
*   2.  Write final output file to ddname MYOUTPUT                *
*   3.  Pack pagedef P1TINLN1 plus any fonts and psegs it         *
*       refrerences.                                              *
*   4.  Pack formdef F1TINLN1 plus any overlays it references, plus *
*       any psegs and fonts (both bounded and unbounded) that those *
*       overlays in turn reference.                               *
*   5.  Pack overly O1TINLN1 plus any psegs that it references.   *
*       Note that an overlay retrieved because of statement 4 will *
*       have both referenced psegs and referenced fonts packed along *
*       with it, because of the NEST= parm on the FORMDEF request. *
*       This request, however, (for overlay O1TINLN1) will only   *
*       retrieve psegs referenced by O1TINLN1 because that is what *
*       is specified on the NEST= parameter of this packing request. *
*   6.  Pack pseg S1TINLN1.                                       *
*   7.  Change the input print file from the default (ILRIN) to   *
*       INPUT2.  This causes the previous request group to be ended, *
*       the print output file to be written, and all files except *
*       ILRSYSIN to be closed.  The program now accepts dd        *
*       redefinition cards until it encounters an object packing  *
*       request.                                                  *
*   8.  Change the output print file from the default (ILROUT) to *
*       OUTPUT2.  This is an object packing request, so all files *
*       are reopened and any new ddnames become active.           *
*   9.  The SCANFILE command causes the entire input print file   *
*       (INPUT2) to be scanned and any psegs or bounded box (i.e. *
*       3820) fonts that it references are packed into the resource *
*       group.                                                    *
*  10.  Pack pagedef P1JUNK plus any psegs and bounded box (i.e.  *
*       3820) fonts that it references.                           *
*  11.  Pack formdef F1TRASH                                     *
```

```
*  12   Change the input print file from INPUT2 to AFPDSIN.  This    *
*       causes the previous request group to be ended, the print     *
*       output file to be written, and all files except ILRSYSIN to  *
*       be closed.                                                   *
*  13.  Change the output print file from OUTPUT2 to OUTPUT3.        *
*  14.  Reopen all files, scan the input print file (AFPDSIN) and    *
*       pack all psegs that it references.                           *
*                                                                    *
*                                                                    *
*   OUTPUT FILE CHARACTERISTICS:                                     *
*   ----------------------------                                     *
*      The output file will have the following characteristics:      *
*          - variable blocked records                                *
*          - carriage control, either the same as the input file or  *
*            ansi if input has no carriage control. If input has no  *
*            carriage control output records will have a blank CC    *
*            prefixed to the beginning of the original record.       *
*          - LRECL will be set to the largest LRECL found in the     *
*            5 resource libraries libraries and the input dataset.   *
*          - BLKSIZE will be set to the largest BLKSIZE found in     *
*            the 5 resource libraries libraries and the input        *
*            dataset                                                 *
*                                                                    *
*      These characteristics are ENFORCED and if an existing file is *
*      referenced by the "ILROUT" DD, it will be OVERWRITTEN with    *
*      these characteristics.                                        *
*                                                                    *
*   GENERAL LOGIC:                                                   *
*   --------------                                                   *
*      Comments in the following code use a convention.  Comment     *
*   blocks that precede a major piece of logic or a subroutine are   *
*   enclosed in asterisks (i.e. *******).  Comment blocks that mark  *
*   a significant point within a piece of inline code are marked by  *
*   hypens (i.e. ------).                                            *
*                                                                    *
*   BASIC LOGIC FLOW IS:                                             *
*                                                                    *
*     1. Main program housekeeping                                   *
*     2. Request group initialization (label READDD). Reads DD redef *
*        cards, opens files for request group, writes BRG record to  *
*        output file.                                                *
*     3. Request processing loop (label READREQ).  Reads packing     *
*        request, if a DD redef go to step 2., otherwise retrieve    *
*        named object from resource library and pack into output     *
*        file.  If internal references are encountered while reading *
*        the resource file and NEST= has been coded, these requests  *
*        are built into an internal request list called NESTLIST.    *
*                                                                    *
*        At EOF on the resource being read, the NESTLIST is checked  *
*        to see if any entries exist.  If so, each entry is processed*
*        as if it had been read from the control file, using the nest*
*        parameter currently in effect from the control file (i.e.   *
*        the references made from within nested objects will be      *
*        handled using the same rules as for the main object.)  When *
*        the NESTLIST is empty, we go back to step 3.                *
*                                                                    *
*        This loop continues until a new request group or EOF on     *
*        ILRSYSIN. At that point an end resource group (ERG) record  *
*        is written to the output file to terminate the resource     *
*        group and the print input file is copied to the print output*
*        file.  At EOF on the ILRIN file, all files except ILRSYSIN  *
*        are closed and and we return to step 2, unless there is also*
*        EOF on ILRSYSIN in which case we terminate.                 *
*                                                                    *
**********************************************************************
*
```

```
*********************************************************************
*             MAIN PROGRAM INITIALIZATION AND HOUSEKEEPING          *
*********************************************************************
         PRINT GEN
         USING *,R15        USE CALLER'S REG. AS TEMPORARY BASE
         STM   R14,12,12(13) SAVE CALLER'S REGS. IN CALLER'S S.AREA
         B     BYLITS
         DC    CL8'ILRPACK '           NAME OF THIS ROUTINE
         DC    CL8'&SYSDATE'           DATE OF THIS ASSEMBLY
BYLITS   EQU   *
         LA    R2,SAVE1     GET OWN S.AREA AND DO FORWARD-
         ST    R2,8(R13)    -POINTING IN CALLER'S SAVEAREA
         ST    R13,SAVE1+4   DO BACKWARD POINTING IN OWN S.AREA
         BAL   R13,START     MAKE OWN SAVEAREA CURRENT & GOTO START
         USING SAVE1,R13
         DROP  R15
SAVE1    DS    18F           STORAGE FOR OWN SAVEAREA
*-------------------------------------------------------------------*
*        CHECK FOR EXEC PARMS, NEW DDNAME FOR SYSIN MAY BE SUPPLIED  *
*-------------------------------------------------------------------*
START    EQU   *
         L     RDA,DATCSECT           |RDA->DATA CSECT
         USING ILRPACKD,RDA           |ESTABLISH ADDRESSABILITY
         L     R1,0(R1)               |R1->INPUT EXEC PARM AREA
         LA    R1,0(R1)               |CLEAR HIGH ORDER BYTE
         CLC   HWNINE,0(R1)           |WAS A PARMLIST PROVIDED?
         BNL   GETBUF                 |BR IF NOT
         CLC   2(9,R1),=C'ILRSYSIN='  |IS PARM ILRSYSIN=?
         BNE   GETBUF                 |BR IF NOT
         MVC   ILRSYSIN+DDNAME(8),=CL8' '  |CLEAR DDNAME FIELD
         LA    R2,ILRSYSIN+DDNAME     |R2->DDNAME IN SYSIN DCB
         LH    R3,0(R1)               |R3 = LENGTH OF PARM FIELD
         SH    R3,HWNINE              |R3 = LEN LESS LEN OF ILRSYSIN=
         LA    R4,8(R1)               |R4 -> NEW DDNAME
         LR    R5,R3                  |R5 = MOVE LENGTH
         MVCL  R2,R4                  |MOVE NEW DDNAME TO ILRSYSIN DCB
*-------------------------------------------------------------------*
*        INITIALIZE MAIN RESOURCE INPUT BUFFER AND PACKWORK AREA     *
*-------------------------------------------------------------------*
GETBUF   EQU   *
         GETMAIN RU,LV=RBUFSIZE       |GET INPUT RESOURCE BUFFER AREA
         LR    RRB,R1                 |RRB -> RESOURCE INPUT BUFFER
         USING RESBUF,RRB             |SET ADDRESSIBILITY
         USING RESTBENT,RLT           |ADDRESSABILITY TO RESLIBTB NTRY
         GETMAIN RU,LV=PKWKSIZE       |GET PACKWORK AREA
         LR    RPW,R1                 |RPW -> PACKWORK AREA
         USING PACKWORK,RPW           |SET ADDRESSIBILITY
         LR    R2,R1                  |R2-> WORKAREA
         L     R3,=AL4(PKWKSIZE-8)    |R3 = LENGTH TO CLEAR
         LA    R4,0                   |SET FOR SHOW, NOT NEEDED
         L     R5,PWCLEAR             |BLANK PAD BYTE, ZERO LENGTH
         MVCL  R2,R4                  |CLEAR PACKWORK TO SPACES
         LA    R1,LISTAREA            |R1 -> START OF PACKLIST
         MVC   PACKWORK(8),=CL8'PACKWORK'   |PUT ID IN AREA
         ST    R1,PAKLSTRT            |SAVE POINTER
         ST    R1,PAKLNEXT            |INDICATE EMPTY LIST
         L     R1,NSTLOFFS            |GET OFFSET TO NESTLIST
         LA    R1,LISTAREA(R1)        |GET ADDRESS OF NESTLIST
         BCTR  R1,R0                  |DECREMENT BY 1
         BCTR  R1,R0                  |DECREMENT BY ANOTHER 1
         ST    R1,PAKLMAX             |STORE MAX EXTENT OF PACKLIST
         LA    R1,LISTAREA            |R1 -> START OF PACKLIST
         L     R2,NSTLOFFS            |R2 -> OFFSET TO NESTLIST
         LA    R1,0(R2,R1)            |R1 -> START OF NESTLIST
         ST    R1,NSTLSTRT            |SAVE START POINTER
         ST    R1,NSTLNEXT            |INDICATE NESTLIST IS EMPTY
```

```
         ST    R1,NSTLCURR            |INDICATE NESTLIST IS EMPTY
         L     R2,NSTLSIZE            |R2 = MAX LENGTH OF NESTLIST
         LA    R1,0(R2,R1)            |R1 -> MAX EXTENT OF NESTLIST
         ST    R1,NSTLMAX             |STORE MAX EXTENT OF NESTLIST
         EXTRACT TIOTADDR,FIELDS=TIOT
         L     R2,TIOTADDR            |R2 -> TIOT
         LA    R2,24(R2)              |R2 -> 1ST TIOT ENTRY
         ST    R2,TIOTADDR            |SAVE POINTER TO FIRST ENTRY
*-------------------------------------------------------------------*
*        OPEN INPUT CONTROL FILE (SYSIN).  THIS IS DONE ONLY ONCE,  *
* ALL OTHER FILES ARE OPENED ONCE FOR EACH REQUEST GROUP ENCOUNTERED.*
*-------------------------------------------------------------------*
OPNSYSIN EQU   *
         MVC   WTOTEXT(8),ILRSYSIN+DDNAME |PUT DDNAME IN MSG
         OPEN  (ILRSYSIN)               |OPEN INPUT CONTROL FILE
         TM    ILRSYSIN+OPENFLAG,DCBOFOPN |TEST FOR SUCCESSFUL OPEN
         BO    READDD                   |BR IF OK, OTHERWISE STOP NOW
         MVC   WTOTEXT+8,=CL50' DD COULD NOT BE OPENED'
         B     ABORT
*
*********************************************************************
*              REQUEST GROUP INTITIALIZATION                        *
*    THE FOLLOWING CODE READS DD REDEFINITION CARDS, THEN OPENS ALL  *
* THE NECESSARY FILES FOR THE REQUEST PROCESSING LOOPS.              *
*********************************************************************
*
*-------------------------------------------------------------------*
*        LOOK FOR DDNAME REDEFINITIONS.                             *
* REDEFINITIONS HAVE THE DEFAULT DDNAME IN COLS 1-8,                *
* AN "=" IN COL 9 AND THE NEW DDNAME IN COLS 10-17.  ALL DD REDEFS  *
* MUST PRECEDE THE FIRST RESOURCE REQUEST CARD FOR THE REQUEST GROUP.*
* THE FIRST SYSIN INPUT RECORD WITHOUT "=" IN COL 9 IS TAKEN AS THE  *
* FIRST RESOURCE REQUEST RECORD.                                     *
*-------------------------------------------------------------------*
READDD   EQU   *
         GET   ILRSYSIN,REQREC        |GET AN INPUT RECORD
         CLI   DDEQUAL,C'='           |IS IT A DDNAME DEFINITION?
         BNE   OPENSYSP               |BR IF NOT, WE HAVE 1ST REQ REC
*
*-------------------------------------------------------------------*
*        SET UP POINTERS TO DDNAME TABLE FOR SEARCH                 *
*-------------------------------------------------------------------*
INITDDTB EQU   *
         LA    R3,DDNAMETB            |R3 -> TABLE OF DDNAMES
         LA    R4,DDNAMELN            |R4 = INCREMENT
         LA    R5,DDNAMETX            |R5 = END OF DDNAME TABLE
*
*-------------------------------------------------------------------*
*        FIND DEFAULT DDNAME (LEFT SIDE OF =) IN DDNAME TABLE       *
*-------------------------------------------------------------------*
FINDDD   EQU   *
         CLC   REQTYPE,0(R3)          |IS IP DDNAME = TABLE DDNAME?
         BE    CHGDDNAM               |IF YES, LEAVE LOOP
         BXLE  R3,R4,FINDDD           |INCR POINTER & LOOP
         LA    R1,WTOLIST             |R1 -> WTO PARM LIST
         MVC   WTOTEXT(8),REQTYPE     |PUT FAILED DDNAME IN MSG
         MVC   WTOTEXT+9,=CL35'INVALID FILE REFERENCE'
         WTO   MF=(E,(1))             |PUT OUT MSG
         B     READDD                 |NOT FOUND, IGNORE DDCARD
CHGDDNAM EQU   *
         L     R4,8(R3)               |R4 -> DCB FOR DDNAME ON CARD
         MVC   DDNAME(8,R4),REQNAME   |PUT NEW DDNAME IN DCB
         B     READDD                 |LOOP TO NEXT CARD
*
*-------------------------------------------------------------------*
*        OPEN ILRSYSPR REPORT FILE                                  *
```

```
*-------------------------------------------------------------------*
OPENSYSP EQU   *
         MVC   WTOTEXT(8),ILRSYSPR+DDNAME  |PUT DDNAME IN MSG TEXT
         MVC   REPDDNAM,ILRSYSPR+DDNAME    |PUT DDNAME IN REP TEXT
         OPEN  (ILRSYSPR,(OUTPUT))    |OPEN ILRSYSPR REPORT FILE
         TM    ILRSYSPR+OPENFLAG,DCBOFOPN |TEST FOR SUCCESSFUL OPEN
         BO    OPNRLIBS               |BR IF OK, OTHERWISE STOP NOW
         MVC   WTOTEXT+9(35),=CL35' REPORT DD COULD NOT BE OPENED'
         B     ABORT
*-------------------------------------------------------------------*
*        OPEN INPUT RESOURCE LIBRARIES                              *
*-------------------------------------------------------------------*
OPNRLIBS EQU   *
         PUT   ILRSYSPR,HYPHNMSG      |WRITE SEPARATOR
         PUT   ILRSYSPR,INITMSG       |WRITE HEADER TO ILRSYSPR
         PUT   ILRSYSPR,INITMSG2      |WRITE HEADER TO ILRSYSPR
         PUT   ILRSYSPR,BLANKMSG      |WRITE SPACER
*        OPEN  (ILRPDEF,,ILRFDEF,,ILRPSEG,,ILROVLY,,ILRFONT)
         LA    RLT,RESLIBTB           |RLT -> TABLE OF RESOURCE LIBS
         LA    R4,RESLIBLN            |R4 = INCREMENT
         LA    R5,RESLIBX1            |R5 = END OF RESOURCE LIB TABLE
*
*-------------------------------------------------------------------*
*        CHECK RESOURCE LIBRARY DCBS TO VERIFY SUCCESSFUL OPEN      *
*-------------------------------------------------------------------*
CHKRLDCB EQU   *
         L     R2,RESTDCB             |R2->DCB
         BAL   RLINK1,SCANTIOT        |CHECK IF DDNAME EXISTS
         C     R15,ZERO               |IS IT OK?
         BNE   WRTDCBEM               |WRITE WARNING MSG IF NOT
         OPEN  ((R2))
         TM    OPENFLAG(R2),DCBOFOPN  |THIS DCB GET OPENED OK?
         BNO   WRTDCBEM               |IF NOT, GO WRITE ERROR MSG
DCBINCR  BXLE  RLT,R4,CHKRLDCB        |INCR POINTER & LOOP
         B     CHKPDFRL               |END OF LIST
WRTDCBEM EQU   *
         MVC   DCBERNAM,DDNAME(R2)    |PUT DDNAME INTO ERROR MESSAGE
         PUT   ILRSYSPR,DCBOPMSG      |WRITE ERROR MESSAGE
         CLI   RCODE,4                |CURRENT RCODE > 4?
         BH    DCBINCR                |RETURN TO LOOP IF YES
         MVI   RCODE,4                |SHOW NON-ZERO RETURN CODE
         B     DCBINCR                |RETURN TO LOOP
*-------------------------------------------------------------------*
*        PLUG HIGEST RESOURCE LRECL AND BLKSIZE INTO ILROUT DCB     *
*-------------------------------------------------------------------*
CHKPDFRL EQU   *
         CLC   ILRPDEF+LRECL,ILROUT+LRECL       |PDEF LRECL > ILROUT?
         BNH   CHKFDFRL               |BRANCH IF NOT
         MVC   ILROUT+LRECL(2),ILRPDEF+LRECL      |COPY PD LRECL TO OP
         MVC   ILROUT+BLKSIZE(2),ILRPDEF+BLKSIZE     |PD BLKSIZE TO OP
CHKFDFRL EQU   *
         CLC   ILRFDEF+LRECL,ILROUT+LRECL       |FDEF LRECL > ILROUT?
         BNH   CHKOVRRL               |BRANCH IF NOT
         MVC   ILROUT+LRECL(2),ILRFDEF+LRECL     |COPY FD LRECL TO OP
         MVC   ILROUT+BLKSIZE(2),ILRFDEF+BLKSIZE |FD BLKSIZE TO OP
CHKOVRRL EQU   *
         CLC   ILROVLY+LRECL,ILROUT+LRECL        |OVLY LRECL > ILROUT?
         BNH   CHKPSGRL               |BRANCH IF NOT
         MVC   ILROUT+LRECL(2),ILROVLY+LRECL     |COPY OV LRECL TO OP
         MVC   ILROUT+BLKSIZE(2),ILROVLY+BLKSIZE |OV BLKSIZE TO OP
CHKPSGRL EQU   *
         CLC   ILRPSEG+LRECL,ILROUT+LRECL       |PSEG LRECL > ILROUT?
         BNH   CHKFNTRL               |BRANCH IF NOT
         MVC   ILROUT+LRECL(2),ILRPSEG+LRECL    |COPY PS LRECL TO OP
         MVC   ILROUT+BLKSIZE(2),ILRPSEG+BLKSIZE |PS BLKSIZE TO OP
CHKFNTRL EQU   *
```

```
         CLC   ILRFONT+LRECL,ILROUT+LRECL      |FONT LRECL > ILROUT?
         BNH   OPENINPT               |BRANCH IF NOT
         MVC   ILROUT+LRECL(2),ILRFONT+LRECL     |COPY FO LRECL TO OP
         MVC   ILROUT+BLKSIZE(2),ILRFONT+BLKSIZE |FO BLKSIZE TO OP
*-------------------------------------------------------------------*
*        OPEN PRINT INPUT FILE. ENSURE LRECL AND BLKSIZE OF ILROUT   *
*   ARE NOT LESS THAN CORRESPONDING ILRIN PARAMETERS.                *
*-------------------------------------------------------------------*
OPENINPT EQU   *
         MVC   PRIPDDNM,ILRIN+DDNAME        |SAVE DDNAME FOR MESSAGES
         OPEN  (ILRIN)                      |OPEN INPUT PRINT FILE
         TM    ILRIN+OPENFLAG,DCBOFOPN      |WAS OPEN OK?
         BO    OPENIPOK                     |BRANCH IF YES
         MVC   WTOTEXT(8),PRIPDDNM          |PUT DDNAME IN MSG
         MVC   WTOTEXT+9(35),=CL35' INPUT FILE DD COULD NOT BE OPENED'
         B     ABORT
OPENIPOK EQU   *
         CLC   ILRIN+LRECL(2),ILROUT+LRECL         |CHECK INPUT LRECL
         BNH   RECLOK                 |INPUT LRECL < CURRENT, BRANCH
         MVC   ILROUT+LRECL(2),ILRIN+LRECL        |COPY IP LRECL TO OP
RECLOK   EQU   *
         CLC   ILRIN+BLKSIZE(2),ILROUT+BLKSIZE    |CHK INPUT BLKSIZE
         BNH   BLKLOK                 |INPUT BLKSZ < CURRENT, BRANCH
         MVC   ILROUT+BLKSIZE(2),ILRIN+BLKSIZE    |IP BLKSIZE TO OP
BLKLOK   EQU   *
         TM    ILRIN+RECFM,DCBRECCA |IS INPUT ANSI CARRIAGE CNTRL?
         BO    CHKOPTCD               |IF YES, GO CHECK OPTCD
         TM    ILRIN+RECFM,DCBRECCM |IS INPUT MACH CARRIAGE CNTRL?
         BO    INMACHCC               |BR IF YES
*-------------------------------------------------------------------*
*    INPUT FILE HAS NO CARRIAGE CONTROL,
*    WE MUST ADD A CHAR TO OUTPUT BLKSIZE AND LRECL TO ACCOMMODATE
*-------------------------------------------------------------------*
         LH    R1,ILROUT+LRECL        |GET ILROUT LRECL
         LA    R1,1(R1)               |BUMP BY 1 TO ADD CC CHARS
         STH   R1,ILROUT+LRECL        |STORE BACK INTO DCB
         LA    R1,4(R1)               |BUMP LRECL BY 4
         CH    R1,ILROUT+BLKSIZE      |IS NEW LRECL+4 > BLKSIZE?
         BNH   CHKOPTCD               |IF NOT, GO CHECK OPTCD
         STH   R1,ILROUT+BLKSIZE      |STORE NEW BLKSIZE IN DCB
         B     CHKOPTCD               |NOW GO CHECK OPTCD
INMACHCC EQU   *
         MVI   ILROUT+RECFM,MACHCC    |SET MACHINE CC ON OUTPUT
*
*-------------------------------------------------------------------*
*        CHECK FOR OPTCD=J IN INPUT FILE AND CARRY THROUGH TO OUTPUT *
*-------------------------------------------------------------------*
CHKOPTCD EQU   *
         TM    ILRIN+OPTCD,DCBOPTJ    |DOES INPUT HAVE TRCS?
         BNO   OPENOUT                |BRANCH IF NOT
         OI    ILROUT+OPTCD,DCBOPTJ |SET OPTCD=J IN OUTPUT
*
*-------------------------------------------------------------------*
*        OPEN PRINT OUTPUT FILE                                     *
*-------------------------------------------------------------------*
OPENOUT  EQU   *
         TM    ILRIN+RECFM,DCBRECV    |IS INPUT RECFM=V
         BO    OPENOUT1               |BR IF YES
         MVC   RDWADJ,ZERO            |SET RDW ADJUSTMENT TO ZERO
*                                      |  - ACCOUNTS FOR FIXED LEN INPUT
OPENOUT1 EQU   *
         MVC   PROPDDNM,ILROUT+DDNAME     |SAVE DDNAME FOR MSGS
         OPEN  (ILROUT,(OUTPUT))      |OPEN PRINT OUTPUT FILE
         TM    ILROUT+OPENFLAG,DCBOFOPN     |WAS OPEN OK?
         BO    WRITEBRG                     |BRANCH IF YES
         MVC   WTOTEXT(8),PROPDDNM          |PUT DDNAME IN MSG
```

```
         MVC   WTOTEXT+9(35),=CL35' OUTPUT FILE DD COULD NOT BE OPENED'
         B     ABORT                        |SCRAM RIGHT NOW
*
*-------------------------------------------------------------------*
*        WRITE A BEGIN RESOURCE GROUP RECORD TO OUTPUT FILE         *
*  INITIALIZE POINTERS IN RESOURCE INPUT BUFFER FOR FIRST READ      *
*-------------------------------------------------------------------*
WRITEBRG EQU   *
         PUT   ILROUT,BRGREC          |WRITE INITIAL BEGIN RES GRP
         LA    R1,RESRDW              |R1 -> 1ST RDW IN BLOCK
         ST    R1,RESRDWP             |SAVE POINTER IN BUFFER AREA
         SLR   R1,R1                  |CLEAR R1
         ST    R1,RESEOBP             |SET END-BLOCK POINTER TO ZERO
         PUT   ILRSYSPR,BLANKMSG      |SPACE REPORT FILE 1
         B     REQ1                   |BYPASS REQ READ, WE HAVE 1ST
*
*********************************************************************
*              END OF REQUEST GROUP INITIALIZATION CODE             *
*********************************************************************
*
*********************************************************************
*                     MAIN REQUEST PROCESSING LOOP                  *
*    PACKING REQUESTS ARE READ FROM SYSIN.  DETECTION OF A DD DEF   *
*    COMMAND TERMINATES THE CURRENT REQUEST GROUP AND CAUSES A RESET *
*    TO PROCESS A NEW REQUEST GROUP.  IF NESTED RESOURCE PACKING HAS *
*    BEEN REQUESTED, THEN PROCESSING THE OBJECT EXPLICITLY REQUESTED *
*    ON THE REQUEST CARD MAY RESULT IN NESTED REQUESTS BEING PLACED  *
*    IN THE NESTLIST.  THESE MUST BE PROCESSED BEFORE THE NEXT REQ   *
*    IS PROCESSED FROM SYSIN.  TO AVOID PACKING OBJECTS MORE THAN    *
*    ONCE, THE PACKLIST AREA IS USED TO RECORD OBJECTS ALREADY PACKED*
*********************************************************************
READREQ  EQU   *
         GET   ILRSYSIN,REQREC        |READ A REQUEST RECORD
         CLI   DDEQUAL,C'='           |IS IT A DDNAME DEFINITION?
         BE    ENDREQ                 |BR IF NOT, AT END OF CURR REQ
REQ1     EQU   *
         MVC   MSGREQ,REQREC          |MOVE REQUEST TO INFO MESSAGE
         MVC   MSGOUT,PROCMSG         |OUTPUT IN PROCESS MSG
         MVC   MSGFLAG,=C' ===>'      |SET UP VISUAL FLAG
         PUT   ILRSYSPR,MSGREC        |WRITE INFO MSG
*        MVC   MSGFLAG,=C'   =>'      |MOVE EXPLICIT REQ FLAG TO MSG
         BAL   RLINK1,PARSNEST        |GO SET NEST FLAGS IF REQD
         CLC   REQTYPE,=CL8'SCANFILE' |REQUEST TO SCAN INPUT FILE?
         BNE   REQ2                   |IF NOT, CHECK FOR NORMAL REQ
         BAL   RLINK1,SCANFILE        |GO SCAN INPUT FILE FOR REFS
         B     REQ3                   |GO HANDLE NESTED REQUESTS
REQ2     EQU   *
         BAL   RLINK1,GETRES          |GO FIND REQUESTED RESOURCE
         C     R15,ZERO               |ARE WE READY TO READ RESOURCE?
         BNE   READREQ                |GET NEXT REQUEST IF NOT
         BAL   RLINK1,PACKOBJ         |GO PACK REQUESTED RESOURCE
REQ3     EQU   *
         TM    NESTFLAG,NESTREQ       |WERE NESTED REQUESTS FOUND?
         BO    PACKNEST               |IF YES, GO PROCESS NESTED REQS
         B     READREQ                |GO READ ANOTHER REQUEST
*
*-------------------------------------------------------------------*
*     PACK OBJECTS FOUND IN THE NESTED REQUEST LIST.  THIS LOOP IS  *
* ENDED WHEN LIST EXHAUSTED, INDICATED BY R2->NEXT AVAILABLE ENTRY  *
*-------------------------------------------------------------------*
PACKNEST EQU   *
         L     R2,NSTLSTRT            |R2->BEGINNING OF NESTED REQ LST
PAKNST1  EQU   *
         C     R2,NSTLNEXT            |IS LIST NOW EMPTY?
         BL    PAKNST2                |BR IF NOT, DO NEXT NEST REQ
         MVC   NSTLNEXT,NSTLSTRT      |RESET NESTLIST TO EMPTY
```

```
         B     READREQ                 |GO GET NEXT USER REQUEST
PAKNST2  EQU   *
         MVC   REQREC(17),0(R2)        |PUT NESTED REQUEST INTO REQ REC
         MVC   REQREC+17(32),BLANKMSG  |CLEAR NEST FLAGS FROM REQ AREA
         LA    R2,WRKLNTLN(R2)         |BUMP PTR TO NEXT LIST ITEM
         MVC   MSGREQ,REQREC           |MOVE REQUEST TO INFO MESSAGE
         MVC   MSGFLAG,NSTMFLAG        |MOVE NESTED REQ FLAG TO MSG
         MVC   MSGFLAG,=C'     +'      |MOVE NESTED REQ FLAG TO MSG
         BAL   RLINK1,GETRES           |GO FIND REQUESTED RESOURCE
         C     R15,ZERO                |ARE WE READY TO READ RESOURCE?
         BNE   PAKNST1                 |GET NEXT REQUEST IF NOT
         BAL   RLINK1,PACKOBJ          |GO PACK NESTED RESOURCE
         B     PAKNST1                 |GO LOOK AT NEXT NESTED REQUEST
*
*-------------------------------------------------------------------*
*     WE GOT AN END-OF-REQUEST GROUP INDICATOR (EITHER EOF ON SYSIN  *
* OR ENCOUNTERED A DD REDEF CARD IN SYSIN)                          *
*-------------------------------------------------------------------*
ENDREQ   EQU   *
         PUT   ILROUT,ERGREC           |WRITE FINAL END RES GRP REC
         L     R1,PACKCTR              |R1 = PACKED COUNT
         LTR   R1,R1                   |WAS ANYTHING PACKED?
         BP    ENDREQ1                 |BR IF YES
         MVI   RCODE,8                 |SET RC FOR POSSIBLE PROBLEM
         PUT   ILRSYSPR,BLANKMSG       |
         PUT   ILRSYSPR,NOPAKMSG       |WRITE WARNING MESSAGE
         PUT   ILRSYSPR,BLANKMSG       |
ENDREQ1  EQU   *
         MVC   COPYIPNM,PRIPDDNM       |PUT INPUT DDNAME IN MSG
         MVC   COPYOPNM,PROPDDNM       |PUT OUTPUT DDNAME IN MSG
         PUT   ILRSYSPR,COPYMSG        |WRITE COPY NOTICE
         PUT   ILRSYSPR,=CL121' '      |WRITE BLANK LINE
*
*-------------------------------------------------------------------*
*     COPY INPUT FILE TO OUTPUT FILE.  OUTPUT IS ALWAYS VARIABLE    *
* LENGTH RECORDS.  LRECL IS CAPTURED FROM INPUT RECORD AND PLACED   *
* IN RDW FOR OUTPUT RECORD.  IF INPUT HAS NO CARRIAGE CONTROL, OUTPUT*
* IS SHIFTED ONE BYTE RIGHT TO MAKE A BLANK IN CC CHARACTER.  OUTPUT *
* MUST HAVE CARRIAGE CONTROL FOR INLINE RESOURCES TO BE DETECTED.   *
* LOOP ENDS AT END-OF-FILE ON ILRIN, WHICH BRANCHES TO EOFPRTIN     *
*-------------------------------------------------------------------*
         MVI   PRTINFLG,COPYFLAG       |SET INPUT COPY FLAG
COPYFILE EQU   *
         GET   ILRIN                   |GET PRINT INPUT RECORD
         LR    R4,R1                   |R4 -> RECORD READ
         LH    R3,ILRIN+LRECL          |R3 = LEN RECORD JUST READ
         AH    R4,RDWADJ               |ADJUST REC PTR FOR RDW
         SH    R3,RDWADJ               |ADJUST REC LEN FOR RDW
         LR    R5,R3                   |R5 ALSO = LEN OF RECORD
         STH   R5,RESRDW               |SAVE REC LENGTH SO FAR
         LA    R2,RESREC               |R2 -> 1ST BYTE OF OUTPUT PRDATA
         TM    ILRIN+RECFM,DCBRECCC    |WAS CC USED ON INPUT?
         BNZ   CLCCOK2                 |BR IF YES
         MVI   0(R2),C' '              |BLANK FIRST CHAR IN OP BUFF
         LA    R2,1(R2)                |BUMP TARGET POINTER BY 1
         LA    R5,1(R5)                |BUMP REC LENGTH BY 1
         STH   R5,RESRDW               |SAVE NEW REC LENGTH
         LR    R5,R3                   |RESET ORIGINAL MOVE LENGTH
CLCCOK2  EQU   *
         MVCL  R2,R4                   |MOVE INPUT BUFFER TO OUTPUT BUF
         LH    R5,RESRDW               |RESTORE ACTUAL REC LEN - RDW
         LA    R5,4(R5)                |ADD 4 TO LEN FOR RDW
         STH   R5,RESRDW               |SAVE RDW FOR PUT
         PUT   ILROUT,RESRDW           |WRITE IT OUT
         B     COPYFILE
*
```

```
*********************************************************************
*             END OF MAIN REQUEST LOOP                              *
*********************************************************************
*
*
*********************************************************************
*-------------------------------------------------------------------*
*             CALLABLE SUBROUTINES                                  *
*-------------------------------------------------------------------*
*********************************************************************
*
*********************************************************************
*     PACK AN OBJECT INTO THE ILROUT FILE.    THIS ROUTINE ASSUMES  *
*  THAT THE GETRES ROUTINE HAS LOCATED THE OBJECT AND SETUP THE DCB  *
*  TO READ THE RESOURCE OBJECT.  RETURN TO CALLER IS VIA RLINK1.     *
*********************************************************************
PACKOBJ  EQU   *
         STM   R1,R5,INTSAVE1         |SAVE CALLER'S REGS
*
         USING WRKLNTRY,R2            |ESTABLISH R2 AS BASE FOR
*                                     |NESTLIST ENTRIES IN CHECK RTNS
         USING AFPDS,R3               |ESTABLISH R3 AS BASE FOR
*                                     |AFPDS RECORDS
         L     R2,PAKLSTRT            |R3 -> START OF PACKLIST
         LA    R4,WRKLNTLN            |R4 = LENGTH OF LIST ENTRIES
         L     R5,PAKLNEXT            |R5 -> NEXT OPEN ENTRY
         S     R5,=F'4'               |R5 -> LAST USED ENTRY
*
*-------------------------------------------------------------------*
*     CHECK TO SEE IF OBJECT HAS ALREADY BEEN PACKED.               *
*-------------------------------------------------------------------*
PACKCHK  EQU   *
         CLC   WRKLNTRY,REQREC        |REQUEST ALREADY PACKED?
         BE    PAKIGNOR               |IGNORE REQUEST IF YES
         BXLE  R2,R4,PACKCHK          |LOOP TO NEXT ENTRY
         MVC   BROBJTYP,RESTTYPC      |SET OBJ TYPE IN BR RECORD
         MVC   BRRNAME,REQNAME        |INSERT RES NAME IN BR REC
         MVC   ERRNAME,REQNAME        |INSERT RES NAME IN ER REC
         PUT   ILROUT,BRREC           |WRITE BR REC
*
*-------------------------------------------------------------------*
*     NOW PACK RESOURCE OBJECT INTO OUTPUT FILE.  RECORD OPCODES ARE *
*  CHECKED FOR RECORDS THAT MIGHT MAKE REFERENCE TO OTHER OBJECTS    *
*  (I.E. NESTED REFERENCES)                                         *
*-------------------------------------------------------------------*
PACKLOOP EQU   *
         BAL   RLINK2,READRES         |GO READ A RESOURCE RECORD
         PUT   ILROUT,(R3)            |WRITE RESOURCE RECORD
         LA    RLINK2,PACKLOOP        |SET RETURN ADDR FOR CHK RTNS
         CLC   7(3,R3),MCF            |IS IT A MCF RECORD?
         BE    CHKMCF                 |BR IF YES
         CLC   7(3,R3),MMO            |IS IT A MMO RECORD?
         BE    CHKMMO                 |BR IF YES
         CLC   7(3,R3),MPS            |IS IT A MPS RECORD?
         BE    CHKMPS                 |BR IF YES
         CLC   7(3,R3),IPS            |IS IT A IPS RECORD?
         BE    CHKIPS                 |BR IF YES
         CLC   7(3,R3),CFI            |IS IT A CFI RECORD?
         BE    CHKCFI                 |BR IF YES
         B     PACKLOOP               |DO IT SOME MORE
*
*-------------------------------------------------------------------*
*     WE GOT A DUPLICATE PACK REQUEST, WE'RE IGNORING IT.           *
*-------------------------------------------------------------------*
PAKIGNOR EQU   *
         LM    R1,R5,INTSAVE1         |RESTORE CALLERS REGS
```

```
         BR    RLINK1                 |RETURN TO CALLER IF OK
*
*-------------------------------------------------------------------*
*     WE GOT AN END-OF-FILE ON RESOURCE MEMBER WE WERE READING      *
*-------------------------------------------------------------------*
EPAKLOOP EQU   *
         PUT   ILROUT,ERREC           |WRITE END RESOURCE REC
         CLC   MSGFLAG,NSTMFLAG       |NESTED PACK REQUEST?
         BNE   EPAKLOP1               |BR IF NOT, NO MESSAGE
         MVC   MSGOUT,NESTMSG         |OUTPUT NESTMSG
         PUT   ILRSYSPR,MSGREC        |WRITE INFO MSG
EPAKLOP1 EQU   *
         L     R1,PACKCTR             |R1 = CURRENT PACKED COUNT
         LA    R1,1(R1)               |INCR COUNT
         ST    R1,PACKCTR             |STORE COUNT
         L     R1,PAKLNEXT            |R1->LAST
         MVC   0(WRKLNTLN,R1),MSGREQ  |SAVE REQUEST IN PACKLIST
         LA    R1,WRKLNTLN(R1)        |BUMP TO NEXT ENTRY
         ST    R1,PAKLNEXT            |SAVE POINTER
         C     R1,PAKLMAX             |ARE WE PAST MAX ALLOWED?
         BNL   KILL1                  |KILL IF YES
         LM    R1,R5,INTSAVE1         |RESTORE CALLERS REGS
         BR    RLINK1                 |RETURN TO CALLER IF OK
KILL1    EQU   *
         WTO   '>>>>>>>>>>>>> ERROR ERROR ERROR <<<<<<<<<<<<<',        X
               ROUTCDE=(11)
         WTO   'EXCEEDED AVAILABLE SPACE IN PACKLIST WORKAREA',        X
               ROUTCDE=(11)
         WTO   '>>>>>>>>>>>>> ERROR ERROR ERROR <<<<<<<<<<<<<',        X
               ROUTCDE=(11)
         ABEND 999
         DROP  R2
         DROP  R3
*
*********************************************************************
*              END OF PACK OBJECT LOOP                              *
*********************************************************************
*
*
*********************************************************************
*      STRUCTURED FIELD CHECKING ROUTINES                           *
* THE FOLLOWING CHK... ROUTINES ARE CALLED WHEN THE INDICATED TYPE   *
* OF AFPDS RECORD IS ENCOUNTERED IN THE INPUT FILE OR ANOTHER RESRCE *
* OBJECT.  IF THE NESTFLAG FOR THE RESOURCE IS ON THEN A PACK REQUEST*
* IS BUILT INTO THE NESTED REQUEST LIST FOR IT, OTHERWISE IT IS      *
* IGNORED.  THESE ROUTINES MAY BE CALLED EITHER BY PACKOBJ OR BY     *
* SCANFILE.                                                          *
*********************************************************************
*
*-------------------------------------------------------------------*
*     KILL ROUTINE IN CASE WE BLOW THE NESTLIST AREA                *
*-------------------------------------------------------------------*
KILL2    EQU   *
         WTO   '>>>>>>>>>>>>> ERROR ERROR ERROR <<<<<<<<<<<<<',        X
               ROUTCDE=(11)
         WTO   'EXCEEDED AVAILABLE SPACE IN NESTLIST WORKAREA',        X
               ROUTCDE=(11)
         WTO   '>>>>>>>>>>>>> ERROR ERROR ERROR <<<<<<<<<<<<<',        X
               ROUTCDE=(11)
         ABEND 999
         USING WRKLNTRY,R2            |ESTABLISH R2 AS BASE FOR
*                                     |NESTLIST ENTRIES IN CHECK RTNS
         USING AFPDS,R3               |ESTABLISH R3 AS BASE FOR
*                                     |AFPDS RECORDS
*
*-------------------------------------------------------------------*
```

```
*     WE READ A MCF RECORD, CHECK TO SEE IF NESTED FONT PACKING IS   *
*   REQUESTED AND, IF SO, ADD FONTS LISTED IN MAP RECORD TO THE      *
*   NESTLIST.  TWO TYPES OF FONTS MAY BE PACKED, BOUNDED BOX FONTS   *
*   AND UNBOUNDED BOX FONTS.  EITHER TYPE, OR BOTH TYPES MAY BE PAKD *
*   DEPENDING UPON WHAT THE USER REQUESTED IN THE NEST= PARAMETER ON *
*   THE PACKING REQUEST RECORD.  CODING BB RETRIEVES BOUNDED BOX     *
*   FONTS, CODING UB RETRIEVES UNBOUNDED BOX FONTS, AND CODING FO    *
*   RETRIEVES BOTH.                                                  *
*------------------------------------------------------------------*
CHKMCF   EQU   *
         TM    NESTFLAG,NESTFO        |DO WE RETRIEVE NESTED FONTS?
         BZR   RLINK2                 |IF NOT, IGNORE MCF
         STM   R1,R5,INTSAVE2         |SAVE CALLERS REGS
         L     R2,NSTLNEXT            |R2->NEXT AVAIL NESTLIST ENTRY
         C     R2,NSTLMAX             |ARE WE POINTING BEYOND ENDLIST?
         BNL   KILL2                  |KILL IF YES
         SLR   R4,R4                  |CLEAR R4
         IC    R4,MCFRGLEN            |R4 = LEN OF REPEAT GROUP
         LR    R5,R3                  |R5->BEGINNING OF RECORD
         AH    R5,AFPDSRDW            |R5->END OF RECORD + 1
         SR    R5,R4                  |R5->1ST BYTE OF LAST RPT GRP
         S     R5,=AL4(MCFRGOFF)      |R5->VIRTUAL ORIGIN OF LAST RGRP
CHKMCFL  EQU   *
         CLI   MCFCFNAM,X'FF'         |IS THERE A CODED FONT NAME?
         BE    CHKMCF1                |BR IF NOT
         TM    NESTFLAG,NESTBB        |BOUNDED BOX FONTS REQUESTED?
         BZ    CHKMCFO                |BR AROUND IF NOT
*                <<< BOUNDED BOX CODED FONT >>>
         MVC   WRKLTYPE(9),=CL9'CFONT' |SET OBJECT TYPE TO CODED FONT
         MVC   WRKLNAME,MCFCFNAM      |PUT CFONTNAME IN NEST REQ
         MVI   WRKLNAME+1,C'0'        |FORCE BOUNDED BOX PREFIX
         LA    R2,WRKLNTLN(R2)        |R2->NEXT AVAIL NESTLIST ENTRY
CHKMCFO  EQU   *
*                <<< UNBOUNDED BOX CODED FONT >>>
         TM    NESTFLAG,NESTUB        |UNBOUNDED BOX FONTS REQUESTED?
         BZ    CHKMCF1                |BR AROUND IF NOT
         MVC   WRKLTYPE(9),=CL9'CFONT' |SET OBJECT TYPE TO CODED FONT
         MVC   WRKLNAME,MCFCFNAM      |PUT CFONTNAME IN NEST REQ
         LA    R2,WRKLNTLN(R2)        |R2->NEXT AVAIL NESTLIST ENTRY
CHKMCF1  EQU    *
*                <<< CODE PAGE >>>
         CLI   MCFCPNAM,X'FF'         |IS THERE A CODE PAGE NAME?
         BE    CHKMCF2                |BR IF NOT
         MVC   WRKLTYPE(9),=CL9'CPAGE' |SET OBJECT TYPE TO CODEPAGE
         MVC   WRKLNAME,MCFCPNAM      |PUT CODEPAGE NAME IN NEST REQ
         LA    R2,WRKLNTLN(R2)        |R2->NEXT AVAIL NESTLIST ENTRY
CHKMCF2  EQU    *
         CLI   MCFCSNAM,X'FF'         |IS THERE A CHAR SET NAME?
         BE    CHKMCF4                |BR IF NOT
         TM    NESTFLAG,NESTBB        |BOUNDED BOX FONTS REQUESTED?
         BZ    CHKMCF3                |BR AROUND IF NOT
*                <<< BOUNDED BOX CHARACTER SET >>>
         MVC   WRKLTYPE(9),=CL9'CHARSET' |SET OBJECT TYPE TO CHARSET
         MVC   WRKLNAME,MCFCSNAM      |PUT CHARSET NAME IN NEST REQ
         MVI   WRKLNAME+1,C'0'        |FORCE BOUNDED BOX PREFIX
         LA    R2,WRKLNTLN(R2)        |R2->NEXT AVAIL NESTLIST ENTRY
CHKMCF3  EQU    *
         TM    NESTFLAG,NESTUB        |UNBOUNDED BOX FONTS REQUESTED?
         BZ    CHKMCF4                |BR AROUND IF NOT
*                <<< UNBOUNDED BOX CHARACTER SET >>>
         MVC   WRKLTYPE(9),=CL9'CHARSET' |SET OBJECT TYPE TO CHARSET
         MVC   WRKLNAME,MCFCSNAM      |PUT CHARSET NAME IN NEST REQ
         LA    R2,WRKLNTLN(R2)        |R2->NEXT AVAIL NESTLIST ENTRY
CHKMCF4  EQU    *
         BXLE  R3,R4,CHKMCFL          |LOOP TO NEXT REPEAT GROUP
         ST    R2,NSTLNEXT            |SAVE POINTER TO NEXT LIST NTRY
```

```
         LM    R1,R5,INTSAVE2         |RESTORE CALLER REGS
         BR    RLINK2                 |RETURN TO MAIN LOOP
*
*-------------------------------------------------------------------*
*     WE READ A MMO RECORD, CHECK TO SEE IF NESTED OVERLAY PACKING IS*
*   REQUESTED AND, IF SO, ADD OVERLAYS LISTED IN THE MAP RECORD TO   *
*   NESTLIST.                                                        *
*-------------------------------------------------------------------*
CHKMMO   EQU   *
         TM    NESTFLAG,NESTOV        |DO WE RETRIEVE NESTED OVS?
         BZR   RLINK2                 |IF NOT, IGNORE MMO
         STM   R1,R5,INTSAVE2         |SAVE CALLER REGS
         L     R2,NSTLNEXT            |R2->NEXT AVAIL NESTLIST ENTRY
         C     R2,NSTLMAX             |ARE WE POINTING BEYOND ENDLIST?
         BNL   KILL2                  |KILL IF YES
         SLR   R4,R4                  |CLEAR R4
         IC    R4,MMORGLEN            |R4 = LEN OF REPEAT GROUP
         LR    R5,R3                  |R5->BEGINNING OF RECORD
         AH    R5,AFPDSRDW            |R5->END OF RECORD + 1
         SR    R5,R4                  |R5->1ST BYTE OF LAST RPT GRP
         S     R5,=AL4(MMORGOFF)      |R5->VIRTUAL ORIGIN OF LAST RGRP
CHKMMOL  EQU   *
         MVC   WRKLTYPE(9),=CL9'OVERLAY'  |SET OBJECT TYPE TO OVERLAY
         MVC   WRKLNAME,MMONAME       |PUT OVLY NAME IN NEST REQ
         LA    R2,WRKLNTLN(R2)        |R2->NEXT AVAIL NESTLIST ENTRY
         BXLE  R3,R4,CHKMMOL          |LOOP TO NEXT REPEAT GROUP
         ST    R2,NSTLNEXT            |SAVE POINTER TO NEXT LIST NTRY
         LM    R1,R5,INTSAVE2         |RESTORE CALLERS REGS
         BR    RLINK2                 |RETURN TO MAIN LOOP
*
*-------------------------------------------------------------------*
*     WE READ A MPS RECORD, CHECK TO SEE IF NESTED PSEG PACKING IS   *
*   REQUESTED AND, IF SO, ADD PSEGS LISTED IN THE MAP RECORD TO      *
*   NESTLIST.                                                        *
*-------------------------------------------------------------------*
CHKMPS   EQU   *
         TM    NESTFLAG,NESTPS        |DO WE RETRIEVE NESTED PSEGS?
         BZR   RLINK2                 |IF NOT, IGNORE MPS
         STM   R1,R5,INTSAVE2         |SAVE CALLER REGS
         L     R2,NSTLNEXT            |R2->NEXT AVAIL NESTLIST ENTRY
         C     R2,NSTLMAX             |ARE WE POINTING BEYOND ENDLIST?
         BNL   KILL2                  |KILL IF YES
         LR    R5,R3                  |R5->BEGINNING OF RECORD
         AH    R5,AFPDSRDW            |R5->END OF RECORD + 1
         BCTR  R5,R0                  |R5->END OF RECORD
         SLR   R4,R4                  |CLEAR R4
         IC    R4,MPSRGLEN            |R4 = LEN OF REPEAT GROUP
CHKMPSL  EQU   *
         MVC   WRKLTYPE(9),=CL9'PAGESEG'  |SET OBJECT TYPE TO PSEG
         MVC   WRKLNAME,MPSNAME       |PUT PSEG NAME IN NEST REQ
         LA    R2,WRKLNTLN(R2)        |R2->NEXT AVAIL NESTLIST ENTRY
         BXLE  R3,R4,CHKMPSL          |LOOP TO NEXT REPEAT GROUP
         ST    R2,NSTLNEXT            |SAVE POINTER TO NEXT LIST NTRY
         LM    R1,R5,INTSAVE2         |RESTORE CALLERS REGS
         BR    RLINK2                 |RETURN TO MAIN LOOP
*
*-------------------------------------------------------------------*
*     WE READ A IPS RECORD, CHECK TO SEE IF NESTED PSEG PACKING IS   *
*   REQUESTED AND, IF SO, ADD PSEGS NAMED IN IPS RECORD TO NESTLIST  *
*-------------------------------------------------------------------*
CHKIPS   EQU   *
         TM    NESTFLAG,NESTPS        |RETRIEVAL OF NESTED PSEGS ON?
         BZR   RLINK2                 |IF NOT, IGNORE IPS
         STM   R1,R5,INTSAVE2         |SAVE CALLER REGS
         L     R2,NSTLNEXT            |R2->NEXT AVAIL NESTLIST ENTRY
         C     R2,NSTLMAX             |ARE WE POINTING BEYOND ENDLIST?
```

```
         BNL   KILL2                  |KILL IF YES
         MVC   WRKLTYPE(9),=CL9'PAGESEG'  |SET OBJECT TYPE TO PSEG
         MVC   WRKLNAME,IPSNAME       |PUT PSEG NAME IN NEST REQ
         LA    R2,WRKLNTLN(R2)        |R2->NEXT AVAIL NESTLIST ENTRY
         ST    R2,NSTLNEXT            |SAVE POINTER TO NEXT ENTRY
         LM    R1,R5,INTSAVE2         |RESTORE CALLERS REGS
         BR    RLINK2                 |RETURN TO MAIN LOOP
*
*-------------------------------------------------------------------*
*     WE READ A CFI RECORD, CHECK TO SEE IF NESTED FONT PACKING IS   *
*   REQUESTED AND, IF SO, ADD CHARSET AND CODEPAGE REFERENCED BY CFI *
*   RECORD TO NESTLIST.                                             *
*-------------------------------------------------------------------*
CHKCFI   EQU   *
         TM    NESTFLAG,NESTFO        |ANY FONT NEST FLAGS ON
         BZR   RLINK2                 |IF NOT, IGNORE CFI
         STM   R1,R5,INTSAVE2         |SAVE CALLER REGS
         L     R2,NSTLNEXT            |R2->NEXT AVAIL NESTLIST ENTRY
         C     R2,NSTLMAX             |ARE WE POINTING BEYOND ENDLIST?
         BNL   KILL2                  |KILL IF YES
         TM    NESTFLAG,NESTBB        |RETRIEVE BOUNDED BOX FONTS?
         BZ    CHKCFI1                |BR IF NOT
         MVC   WRKLTYPE(9),=CL9'CHARSET'  |SET OBJECT TYPE TO CSET
         MVC   WRKLNAME,CFICSNAM      |PUT CSET NAME IN NEST REQ
         MVI   WRKLNAME+1,C'0'        |FORCE BOUNDED BOX PREFIX
         LA    R2,WRKLNTLN(R2)        |R2->NEXT AVAIL NESTLIST ENTRY
         MVC   WRKLTYPE(9),=CL9'CPAGE' |SET OBJECT TYPE TO CODEPAGE
         MVC   WRKLNAME,CFICPNAM      |PUT CPAGE NAME IN NEST REQ
         LA    R2,WRKLNTLN(R2)        |R2->NEXT AVAIL NESTLIST ENTRY
CHKCFI1  EQU   *
         TM    NESTFLAG,NESTUB        |RETRIEVE UNBOUNDED BOX FONTS?
         BZ    CHKCFI2                |BR IF NOT
         MVC   WRKLTYPE(9),=CL9'CHARSET'  |SET OBJECT TYPE TO CSET
         MVC   WRKLNAME,CFICSNAM      |PUT CSET NAME IN NEST REQ
*                                    | (NOTE THAT UNBOUNDED BOX
*                                    |  PREFIX IS LEFT ALONE IN CASE
*                                    |  A NON-ZERO ROTATION HAS BEEN
*                                    |  SPECIFIED BY APPLICATION)
         LA    R2,WRKLNTLN(R2)        |R2->NEXT AVAIL NESTLIST ENTRY
         MVC   WRKLTYPE(9),=CL9'CPAGE' |SET OBJECT TYPE TO CODEPAGE
         MVC   WRKLNAME,CFICPNAM      |PUT CPAGE NAME IN NEST REQ
         LA    R2,WRKLNTLN(R2)        |R2->NEXT AVAIL NESTLIST ENTRY
CHKCFI2  EQU   *
         ST    R2,NSTLNEXT            |SAVE POINTER TO NEXT ENTRY
         LM    R1,R5,INTSAVE2         |RESTORE CALLERS REGS
         BR    RLINK2                 |RETURN TO MAIN LOOP
         DROP  R2
         DROP  R3
*
*********************************************************************
*              END OF STRUCTURED FIELD CHECKING ROUTINES            *
*********************************************************************
*
*********************************************************************
*     WE GOT AN END-OF-FILE ON INPUT REQUEST FILE (SYSIN)           *
*********************************************************************
EOFSYSIN EQU   *
         MVI   SYSINEOF,X'FF'         |INDICATE RECEIVED EOF ON SYSIN
         B     ENDREQ                 |GO HANDLE END OF REQUEST GROUP
*
*********************************************************************
*              END OF EOFSYSIN                                      *
*********************************************************************
*
*********************************************************************
*  GOT AN END-OF-FILE ON CURRENT PRINT INTPUT FILE.  DETERMINE      *
```

```
*  WHETHER FILE WAS BEING SCANNED FOR RESOURCE REFERENCES OR WAS     *
*  BEING COPIED TO OUTPUT FILE AND TAKE APPROPRIATE ACTION           *
*********************************************************************
EOFPRTIN EQU   *
         CLI   PRTINFLG,SCANFLAG      |SCANNING FOR RESOURCE REFS?
         BNE   EOFPRTI1               |BR IF NOT
         CLOSE (ILRIN)                |CLOSE ILRIN FILE
         OPEN  (ILRIN,(INPUT))        |REOPEN ILRIN FILE
         MVI   PRTINFLG,COPYFLAG      |SET INPUT COPY FLAG
         B     EOFSCAN                |RETURN TO SCANFILE ROUTINE
*-------------------------------------------------------------------*
*   WE FINISHED COPYING THE INPUT FILE, REQUEST IS NOW COMPLETE     *
*   CLOSE ALL FILES EXCEPT SYSIN AND LOOK FOR NEXT REQUEST          *
*-------------------------------------------------------------------*
EOFPRTI1 EQU   *
         CLOSE (ILRPDEF,,ILRFDEF,,ILROVLY,,ILRPSEG,,ILRFONT)
         CLOSE (ILRSYSPR,,ILRIN,,ILROUT)
         MVC   ILRIN+DDNAME(8),=CL8'UNDEFIND' |RESET FOR NEW DDNAME
         MVC   ILROUT+DDNAME(8),=CL8'UNDEFIND' |RESET FOR NEW DDNAME
         CLI   SYSINEOF,X'FF'         |NO MORE REQUESTS?
         BE    GETOUT                 |EXIT IF NOT
         LA    R1,LISTAREA            |R1 -> START OF PACKLIST
         L     R2,NSTLOFFS            |R2 -> OFFSET TO NESTLIST
         LA    R1,0(R2,R1)            |R1 -> START OF NESTLIST
         ST    R1,NSTLSTRT            |SAVE START POINTER
         ST    R1,NSTLNEXT            |INDICATE NESTLIST IS EMPTY
         ST    R1,NSTLCURR            |INDICATE NESTLIST IS EMPTY
         LA    R1,0(R2,R1)            |R1 -> MAX EXTENT OF NESTLIST
         ST    R1,NSTLMAX             |STORE MAX EXTENT OF NESTLIST
         MVI   PRTINFLG,COPYFLAG      |RESET INPUT FILE FLAG
         B     INITDDTB               |GO PROCESS NEW DD DEF CARDS
*
*********************************************************************
*              END OF EOFPRTIN                                      *
*********************************************************************
*
*********************************************************************
*  READ A RESOURCE RECORD FROM THE LIBRARY MEMBER                   *
*     RETURN TO CALLER VIA RLINK2                                   *
*********************************************************************
READRES  EQU   *
         STM   R1,R5,INTSAVE2         |SAVE CALLER'S REGS
         LM    R3,R4,RESBUF           |GET BUFFER CONTROL INFO
*                                     |    R3->START OF NEXT RDW
*                                     |    R4->END OF BLOCK
         CR    R3,R4                  |IS NEXT < END?
         BNL   NEWBLOCK               |GET NEW BLOCK IF NOT
         LH    R4,0(R3)               |R4 = LEN OF CURR BLOCK
         LA    R4,0(R3,R4)            |R4 -> NEXT RDW
         ST    R4,RESRDWP             |SAVE ADDR OF NEXT RDW
         BR    RLINK2                 |RETURN WITH R3->CURR RECORD
NEWBLOCK EQU   *
         LA    R2,RESBDW              |R2 -> INPUT BUFFER
         READ  READDECB,SF,(RESDCB),(R2),'S'
         CHECK READDECB
         LH    R3,RESBDW              |R3 = LENGTH OF NEW BLOCK
         LA    R4,RESBDW(R3)          |R4 ->END OF NEW BLOCK
         LA    R3,RESRDW              |R3 ->FIRST RDW
         STM   R3,R4,RESBUF           |SAVE POINTERS
         B     READRES                |NOW PROCESS NEW BLOCK
*
*********************************************************************
*              END OF READRES                                       *
*********************************************************************
*
*********************************************************************
```

```
*        ROUTINE TO SCAN THE TIOT AND SEE IF DDNAMES EXIST OR NOT.   *
*  R2 IS EXPECTED TO POINT TO THE DCB TO BE CHECKED.                 *
**********************************************************************
SCANTIOT EQU   *
         STM   R1,R5,INTSAVE1          |SAVE CALLER REGS
         L     R3,TIOTADDR             |R3->TIOT FIRST DD
         LA    R15,0                   |SET GOOD RETURN CODE AS DEFAULT
SCANTIOL EQU   *
         CLC   DDNAME(8,R2),4(R3)      |DCB DDNAME IN TIOT?
         BE    SCANTIOX                |EXIT IF YES
         LA    R3,TIOTLEN(R3)          |BUMP POINTER
         CLC   4(4,R3),ZERO            |DDNAME HEX ZEROS?
         BNE   SCANTIOL                |IF NOT, LOOP TO NEXT TIOT ENTRY
         LA    R15,20                  |SET NOFIND RETURN CODE
SCANTIOX EQU   *
         LM    R1,R5,INTSAVE1          |RESTORE CALLERS REGS
         BR    RLINK1                  |RETURN TO CALLER
*
**********************************************************************
*              END OF SCANTIOT                                       *
**********************************************************************
*
**********************************************************************
*        ROUTINE TO PARSE THE NEST= PARAMETER ON THE INPUT CARD AND  *
*   SET NESTFLAG ACCORDINGLY.                                        *
*   RETURN VIA RLINK1                                                *
**********************************************************************
PARSNEST EQU   *
         STM   R1,R5,INTSAVE1          |SAVE CALLER REGS
         NI    NESTFLAG,X'00'          |TURN ALL NEST FLAGS OFF
         CLC   NESTKEYW,=C'NEST='      |NESTED RESOURCE RETRIEVAL REQ?
         BNER  RLINK1                  |IF NOT, RETURN TO CALLER
         LA    R4,3                    |R4 = INCREMENT
         LA    R5,NESTVAL4             |R5->LAST POSSIBLE VALUE
         LA    R2,NESTVALU             |R2->FIRST VALUE
PARSLOOP EQU   *
         CLI   0(R2),C' '              |ITEM BLANK?
         BE    PARSLXIT                |EXIT IF YES
         CLC   0(2,R2),=C'OV'          |RETRIEVE NESTED OVERLAYS?
         BNE   PARSLPS                 |IF NOT, CHK PSEGS
         OI    NESTFLAG,NESTOV+NESTREQ |SET RETRIEVE OVERLAY FLAG
         B     PARSITER                |GO ITERATE LOOP
PARSLPS  EQU   *
         CLC   0(2,R2),=C'PS'          |RETRIEVE NESTED PSEGS?
         BNE   PARSLBB                 |IF NOT, CHK BB FONTS
         OI    NESTFLAG,NESTPS+NESTREQ  |SET RETRIEVE PSEGS FLAG
         B     PARSITER                |GO ITERATE LOOP
PARSLBB  EQU   *
         CLC   0(2,R2),=C'BB'          |RETRIEVE BOUNDED BOX FONTS?
         BNE   PARSLUB                 |IF NOT, CHK UB FONTS
         OI    NESTFLAG,NESTBB+NESTREQ |SET RETRIEVE PSEGS FLAG
         B     PARSITER                |GO ITERATE LOOP
PARSLUB  EQU   *
         CLC   0(2,R2),=C'UB'          |RETRIEVE UNBOUNDED BOX FONTS?
         BNE   PARSLFO                 |IF NOT, CHK ALL FONTS
         OI    NESTFLAG,NESTUB+NESTREQ |SET RETRIEVE PSEGS FLAG
         B     PARSITER                |GO ITERATE LOOP
PARSLFO  EQU   *
         CLC   0(2,R2),=C'FO'          |RETRIEVE ALL NESTED FONTS?
         BNE   PARSITER                |IF NOT, ITERATE LOOP
         OI    NESTFLAG,NESTFO+NESTREQ |SET RETRIEVE ALL FONTS FLAG
PARSITER EQU   *
         BXLE  R2,R4,PARSLOOP          |INCR POINTER AND LOOP
PARSLXIT EQU   *
         LM    R1,R5,INTSAVE1          |RESTORE CALLERS REGS
```

```
         BR    RLINK1                 |RETURN TO CALLER
*
*************************************************************************
*              END OF PARSNEST                                          *
*************************************************************************
*
*************************************************************************
*               SCAN INPUT FILE FOR NESTED OBJECT REFERENCES            *
*   THIS ROUTINE ASSUMES THAT ILRIN IS ALREADY OPEN.    WHEN EOF IS   *
* ENCOUNTERED ON ILRIN THE EOFPRTIN ROUTINE IS CALLED.    THE SETTING*
* OF THE PRTINFLG INDICATES TO EOFPRTIN THAT THE ROUTINE DOING THE    *
* READING WAS THE SCANFILE ROUTINE, NOT THE COPYFILE ROUTINE.         *
* EACH RECORD READ IS CHECKED TO SEE IF IT IS A MAP CODED FONT, MAP   *
* PAGE SEGMENT, OR INCLUDE PAGE SEGMENT STRUCTURED FIELD (THESE ARE   *
* THE ONLY NESTED REFERENCES POSSIBLE FROM A PRINT INPUT FILE).  IF   *
* ONE OF THESE IS FOUND THE APPROPRIATE CHK... ROUTINE IS CALLED TO   *
* HANDLE IT.                                                          *
*************************************************************************
SCANFILE EQU   *
         STM   R1,R5,INTSAVE1         |SAVE CALLER'S REGS
         MVI   PRTINFLG,SCANFLAG      |SET INPUT SCAN FLAG
SCANLP1  EQU   *
         GET   ILRIN                  |GET PRINT INPUT RECORD
         LR    R3,R1                  |R3 -> RECORD READ
         TM    ILRIN+RECFM,DCBRECV    |IS INPUT RECFM=V
         BO    SLRECV                 |BR IF YES
         S     R3,=AL4(4)             |PRETEND AN RDW EXISTS
SLRECV   EQU   *
         CLI   4(R3),X'5A'            |IS RECORD A STR FLD REC?
         BNE   SCANLP1                |LOOP IF NOT
         LA    RLINK2,SCANLP1         |SET RETURN ADDR FOR CHK RTNS
         CLC   7(3,R3),MCF            |IS IT A MCF RECORD?
         BE    CHKMCF                 |BR IF YES
         CLC   7(3,R3),MPS            |IS IT A MPS RECORD?
         BE    CHKMPS                 |BR IF YES
         CLC   7(3,R3),IPS            |IS IT A IPS RECORD?
         BE    CHKIPS                 |BR IF YES
         B     SCANLP1                |LOOP TO NEXT RECORD
EOFSCAN  EQU   *
         LM    R1,R5,INTSAVE1         |RESTORE CALLER REGS
         BR    RLINK1                 |RETURN TO CALLER
*
*************************************************************************
*              END OF SCANFILE                                          *
*************************************************************************
*
*************************************************************************
*        ROUTINE TO VALIDATE A RESOURCE PACK REQUEST, FIND THE MEMBER*
*   IN THE RESOURCE LIBRARY, AND SETUP TO READ THE MEMBER. RETURN    *
*   VIA RLINK1.                                                       *
*************************************************************************
GETRES   EQU   *
         STM   R1,R5,INTSAVE1         |SAVE CALLER'S REGS
         LA    RLT,RESLIBTB           |RLT -> TABLE OF RESOURCE LIBS
         LA    R4,RESLIBLN            |R4 = INCREMENT
         LA    R5,RESLIBTX            |R5 = END OF RESOURCE LIB TABLE
*
*-----------------------------------------------------------------------*
*        FIND OBJECT TYPE IN RESOURCE TABLE (RESLIBTB)                  *
*-----------------------------------------------------------------------*
FINDTYPE EQU   *
         CLC   REQTYPE,RESTTYPL       |IS REQUESTED TYPE = LIB TYPE?
         BE    FINDMBR                |IF YES, LEAVE LOOP
         BXLE  RLT,R4,FINDTYPE        |INCR POINTER & LOOP
         LA    R2,ERRMSG1             |NOT FOUND, SET ERROR MESSAGE
```

```
         B     WRFNDERR               |GO WRITE ERROR MESSAGE
*
*-------------------------------------------------------------------*
*        WE KNOW TYPE OF OBJECT, SEE IF NAMED OBJECT EXISTS IN RESLIB*
*-------------------------------------------------------------------*
FINDMBR  EQU   *
         L     RESDCB,RESTDCB         |RESDCB->LIB TO PULL RES FROM
         TM    OPENFLAG(RESDCB),DCBOFOPN    |IS THIS DCB OPEN?
         BNO   FINDERR                |ISSUE ERROR MSG IF NOT
         FIND  (RESDCB),REQNAME,D     |DO FIND ON REQUESTED MEMBER
         C     R15,ZERO               |DID FIND GIVE ZERO RC?
         BNE   FINDERR                |IF NOT, ISSUE ERROR
         SLR   R15,R15                |RETURN CODE ZERO TO CALLER
         LM    R1,R5,INTSAVE1         |RESTORE CALLER'S REGS
         BR    RLINK1                 |RETURN TO CALLER
FINDERR  EQU   *
         LA    R2,ERRMSG2             |BLDL FAILED, SET ERROR MSG
         B     WRFNDERR               |GO WRITE ERROR MSG
WRFNDERR EQU   *
         MVC   MSGOUT,0(R2)           |PUT MSG TEXT IN OUTPUT AREA
         PUT   ILRSYSPR,MSGREC        |WRITE INFO MSG
         LA    R15,4                  |RETURN CODE 4 TO CALLER
         LM    R1,R5,INTSAVE1         |RESTORE CALLER'S REGS
         BR    RLINK1                 |RETURN TO CALLER
*
*********************************************************************
*              END OF GETRES                                        *
*********************************************************************
*
*********************************************************************
*  EXIT PROGRAM WITH AN ERROR WRITTEN TO PROGRAMMER LOG             *
*********************************************************************
ABORT    EQU   *
         LA    R1,WTOLIST
         WTO   MF=(E,(1))
         MVC   WTOTEXT,=CL50'    >>> EXECUTION ABORTED <<<'
         LA    R1,WTOLIST
         WTO   MF=(E,(1))
         MVI   RCODE,20
*
*********************************************************************
*  EXIT LINKAGE                                                     *
*********************************************************************
GETOUT   EQU   *
         CLOSE (ILRSYSIN)
         FREEMAIN RU,LV=RBUFSIZE,A=(RRB) |FREE BUFFER RESOURCE BUFFER
         FREEMAIN RU,LV=PKWKSIZE,A=(RPW) |FREE BUFFER PACKWORK BUFFER
         CLI   RCODE,4                |WHAT IS MAX RETURN CODE SO FAR?
         BH    SETRCODE               |BR IF HIGHER THAN 4
         L     R1,PACKCTR             |GET PACKED COUNT
         LTR   R1,R1                  |TEST FOR COUNT > 0
         BP    SETRCODE               |BR IF > 0
         MVI   RCODE,8                |SET RCODE TO MEAN NOTHING PACKD
SETRCODE LA    R15,0                  |SET RETURN CODE
RCODE    EQU   SETRCODE+3
         L     R13,4(,R13)            |RESTORE CALLERS SAVE AREA ADDR.
         L     R14,12(,R13)           |RESTORE CALLERS RETURN ADDRESS
         LM    R0,R12,20(R13)         |RESTORE CALLERS REGISTERS
         BR    R14                    |RETURN TO CALLER
*
DATCSECT DC    A(ILRPACKD)
*
*
*********************************************************************
*-------------------------------------------------------------------*
*              DATA AREA DEFINITIONS                                *
```

```
*-------------------------------------------------------------------*
*********************************************************************
*
ILRPACKD CSECT
         DC    CL8'ILRPACKD'
*
*********************************************************************
*        REGISTER EQUATES                                           *
*********************************************************************
R0       EQU   0
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9
R10      EQU   10
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
RDA      EQU   R6       |POINTER TO DATA CSECT
RPW      EQU   R7       |POINTER TO PACKWORK AREA
RLINK1   EQU   R8       |INTERNAL LINKAGE REGISTER
RESDCB   EQU   R9       |POINTER TO CURRENT RESOURCE LIB DCB
RLINK2   EQU   R10      |INTERNAL LINKAGE REGISTER
RLT      EQU   R11      |POINTS TO CURRENT ENTRY IN RES LIB TABLE
RRB      EQU   R12      |BASE FOR RESOURCE INPUT BUFFER
*                                                                   *
RECFM    EQU   DCBRECFM-IHADCB  |OFFSET TO RECFM IN DCB
LRECL    EQU   DCBLRECL-IHADCB  |OFFSET TO LRECL IN DCB
BLKSIZE  EQU   DCBBLKSI-IHADCB  |OFFSET TO BLKSIZE IN DCB
OPTCD    EQU   DCBOPTCD-IHADCB  |OFFSET TO OPTCD IN DCB
DDNAME   EQU   DCBDDNAM-IHADCB  |OFFSET TO DDNAME IN DCB
OPENFLAG EQU   DCBOFLGS-IHADCB  |OFFSET TO OPEN FLAGS IN DCB
RBUFSIZE EQU   32760            |RESOURCE BUFFER AREA IS 32K
PKWKSIZE EQU   64*1024          |PACKWORK AREA IS 64K
PAKLSIZE EQU   15*1024          |PACKLIST SIZE IS 15K
TIOTLEN  EQU   20               |LENGTH OF A TIOT ENTRY
MACHCC   EQU   B'01010010'      |DCB RECFM FOR MACHINE CC
         DC    CL8'INTSAVE1'
INTSAVE1 DC    12F'0'
         DC    CL8'INTSAVE2'
INTSAVE2 DC    12F'0'
TIOTADDR DC    F'0'
ZERO     DC    F'0'
PACKCTR  DC    F'0'
NSTLOFFS DC    AL4(PAKLSIZE+16)
NSTLSIZE DC    AL4(PKWKSIZE-(PAKLSIZE+36))
PWCLEAR  DC    XL1'40',AL3(0)
HWNINE   DC    H'9'
RDWADJ   DC    H'4'             |ADJUSTMENT VALUE FOR RDW
*                               |  0=RECFM=F, 4=RECFM=V
PRIPDDNM DC    CL8' '
PROPDDNM DC    CL8' '
SYSINEOF DC    X'00'
PRTINFLG DC    X'00'
SCANFLAG EQU   X'FF'            |INPUT FILE SCAN IN PROGRESS
COPYFLAG EQU   X'00'            |INPUT FILE COPY IN PROGRESS
NESTFLAG DC    X'00'
NESTOV   EQU   B'00000001'      |RETRIEVE NESTED OVERLAYS
NESTPS   EQU   B'00000010'      |RETRIEVE NESTED PAGESEGMENTS
```

```
NESTBB   EQU   B'00000100'      |RETRIEVE NESTED BOUNDED BOX FONTS
NESTUB   EQU   B'00001000'      |RETRIEVE NESTED UNBOUNDED BOX FONTS
NESTFO   EQU   B'00001100'      |RETRIEVE NESTED FONTS (BOTH TYPES)
NESTREQ  EQU   B'10000000'      |SOME NEST REQUEST WAS MADE
*********************************************************************
*  RESOURCE LIBRARY TABLE
*     ENTRIES MAPPED BY DSECT RESTBENT
*     USED FOR TWO PURPOSES:  RELATE AN OBJECT TYPE TO A DCB ADDR,
*     AND AS A LIST OF RESOURCE LIB DCB ADDRESSES.  RESLIBX1 IS USED
*     AS END-OF-LIST WHEN SCANNING DCB ADDRESSES, RESLIBTX IS USED
*     WHEN RELATING OBJECT TYPES TO DCB ADDRESSES.
*********************************************************************
RESLIBTB DS    0F
         DC    CL8'PAGEDEF ',A(ILRPDEF),A(BRPDEF)
         DC    CL8'FORMDEF ',A(ILRFDEF),A(BRFDEF)
         DC    CL8'OVERLAY ',A(ILROVLY),A(BROVLY)
         DC    CL8'PAGESEG ',A(ILRPSEG),A(BRPSEG)
RESLIBX1 EQU   *
         DC    CL8'CHARSET ',A(ILRFONT),A(BRCSET)
         DC    CL8'CPAGE   ',A(ILRFONT),A(BRCPAGE)
         DC    CL8'CFONT   ',A(ILRFONT),A(BRCFONT)
RESLIBLN EQU   (*-RESLIBTB)/7
RESLIBTX EQU   *-RESLIBLN
*********************************************************************
*  DDNAME TABLE
*********************************************************************
DDNAMETB DS    0F
         DC    CL8'ILRPDEF ',A(ILRPDEF)
         DC    CL8'ILRFDEF ',A(ILRFDEF)
         DC    CL8'ILROVLY ',A(ILROVLY)
         DC    CL8'ILRPSEG ',A(ILRPSEG)
         DC    CL8'ILRFONT ',A(ILRFONT)
         DC    CL8'ILRIN ',A(ILRIN)
         DC    CL8'ILROUT',A(ILROUT)
         DC    CL8'ILRSYSPR',A(ILRSYSPR)
DDNAMELN EQU   (*-DDNAMETB)/8
DDNAMETX EQU   *-DDNAMELN
*********************************************************************
*  INPUT REQUEST RECORD AREA
*********************************************************************
REQREC   DS    0CL80
REQTYPE  DC    CL8' '      |TYPE OF OBJECT: PD,FD,OV,PS,CF,CS,CP
DDEQUAL  DC    CL1' '      |= SIGN FOR DD DEFN RECORDS
REQNAME  DC    CL8' '      |FULL MEMBER NAME OF OBJECT IN RESOURCE LIB
         DC    CL1' '
NESTKEYW DC    CL5' '      |EITHER BLANK OR NEST=
NESTVALU DS    0CL8        |PS,FO,OV,BB,UB (ANY ORDER OR COMBO)
NESTVAL1 DC    CL2' ',C','
NESTVAL2 DC    CL2' ',C','
NESTVAL3 DC    CL2' ',C','
NESTVAL4 DC    CL2' ',C','
         DC    CL50' '
*********************************************************************
INITMSG  DS    0CL121
         DC    CL10' '
         DC    CL46'**** ILRPACK ACTION REPORT WRITTEN TO DDNAME: '
REPDDNAM DC    CL8' '
         DC    CL59' ****             ASSEMBLED: &SYSDATE &SYSTIME'
INITMSG2 DS    0CL121
         DC    CL7' '
         DC    CL114'(''===>'' MEANS EXPLICIT PACKING REQUEST, ''+'' MEX
               ANS NESTED REQUEST)'
NSTMFLAG DC    C'    +'
MSGREC   DS    0CL121
MSGFLAG  DC    CL5' '
MSGREQ   DC    CL35' '
```

```
MSGOUT   DC    CL98' '
NESTMSG  DC    CL98' PACKED DUE TO INDIRECT REFERENCE'
BLANKMSG DC    CL121' '
HYPHNMSG DS    OCL121
         DC    CL1'1'
         DC    CL40'----------------------------------------'
         DC    CL40'----------------------------------------'
         DC    CL40'----------------------------------------'
PROCMSG  DC    CL98' NOW BEING PROCESSED'
         DC    CL76' '
COPYMSG  DS    OCL121
         DC    CL35'  RESOURCE GROUP WRITTEN TO DDNAME '
COPYOPNM DC    CL8' '
         DC    CL25' NOW COPYING FROM DDNAME '
COPYIPNM DC    CL8' '
         DC    CL50' '
ERRMSG1  DC    CL98' >>> INVALID RESOURCE TYPE, NOT PACKED'
ERRMSG2  DC    CL98' >>> NOT FOUND IN LIBRARY, NOT PACKED'
NOPAKMSG DC    CL121' >>> WARNING: PRECEEDING RESOURCE GROUP IS EMPTY, X
               NOTHING WAS PACKED <<<'
DCBOPMSG DS    OCL121
         DC    CL29' >>>> COULD NOT OPEN DDNAME: '
DCBERNAM DC    CL8' '
         DC    CL90'   RESOURCES IN THIS LIBRARY NOT AVAILABLE >>>'
*********************************************************************
WTOLIST  WTO   'ILR:>                                                ',*
               MF=L,ROUTCDE=(11)
WTOLISTL EQU   *-WTOLIST
         ORG   WTOLIST+9
WTOTEXT  DS    CL51
         ORG
*********************************************************************
BLDLLIST DS    0D
         DC    H'1',H'14'
BLDLNAME DC    CL8' '
BLDLFLAG DC    XL6'000000000000'
*                                                                   *
*********************************************************************
*      AFPDS RECORD AREAS                                           *
*********************************************************************
MCF      DC    XL3'D3B18A'    |MAP CODED FONT OP CODE
MMO      DC    XL3'D3B1DF'    |MAP MEDIUM OVERLAY OP CODE
MPS      DC    XL3'D3B15F'    |MAP PAGE SEGMENT OP CODE
IPS      DC    XL3'D3AF5F'    |INCLUDE PAGE SEGMENT OP CODE
CFI      DC    XL3'D38C8A'    |CODED FONT INDEX OP CODE
*
* ------------------BEGIN RESOURCE GROUP STRUCTURED FIELD RECORD    *
BRGREC   DS    0H      BEGIN RESOURCE GROUP RECORD
         DC    AL2(BRGLEN),H'00'
         DC    XL9'5A0008D3A8C6000000'
BRGLEN   EQU   *-BRGREC
* ------------------BEGIN RESOURCE STRUCTURED FIELD RECORD          *
BRREC    DS    0H      |BEGIN RESOURCE RECORD
         DC    AL2(BRLEN),H'00'
BRINTRO  DC    XL9'5A001AD3A8CE000000'    |INTRODUCER
BRRNAME  DC    CL8' '                     |RESOURCE NAME
         DC    XL4'00000821'              |CONSTANT FLAGS
BROBJTYP DC    XL1'00'    |OBJECT TYPE FLAG, WILL BE ONE OF:
BRCSET   EQU   X'40'      |   CHARACTER SET FLAG
BRCPAGE  EQU   X'41'      |   CODE PAGE FLAG
BRCFONT  EQU   X'42'      |   CODED FONT FLAG
BRPSEG   EQU   X'FB'      |   PAGE SEGMENT FLAG
BROVLY   EQU   X'FC'      |   OVERLAY FLAG
BRPDEF   EQU   X'FD'      |   PAGEDEF FLAG
BRFDEF   EQU   X'FE'      |   FORMDEF FLAG
```

```
         DC    XL5'00'    |CONSTANT
BRLEN    EQU   *-BRREC
* ------------------END   RESOURCE STRUCTURED FIELD RECORD          *
ERREC    DS    0H      END RESOURCE RECORD
         DC    AL2(ERLEN),H'0'
ERINTRO  DC    XL9'5A0010D3A9CE000000'    |INTRODUCER
ERRNAME  DC    CL8' '                     |RESOURCE NAME
ERLEN    EQU   *-ERREC
* ------------------END RESOURCE GRP STRUCTURED FIELD RECORD        *
ERGREC   DS    0H      |BEGIN RESOURCE GROUP RECORD
         DC    AL2(ERGLEN),H'00'
         DC    XL9'5A0008D3A9C6000000'
ERGLEN   EQU   *-ERGREC
*                                                                    *
*********************************************************************
*  DCB DEFINITIONS
*********************************************************************
ILRPDEF  DCB   MACRF=R,DSORG=PO,RECFM=VBM,DDNAME=ILRPDEF,             X
               EODAD=EPAKLOOP,BUFL=32760
ILRFDEF  DCB   MACRF=R,DSORG=PO,RECFM=VBM,DDNAME=ILRFDEF,             X
               EODAD=EPAKLOOP,BUFL=32760
ILROVLY  DCB   MACRF=R,DSORG=PO,RECFM=VBM,DDNAME=ILROVLY,             X
               EODAD=EPAKLOOP,BUFL=32760
ILRPSEG  DCB   MACRF=R,DSORG=PO,RECFM=VBM,DDNAME=ILRPSEG,             X
               EODAD=EPAKLOOP,BUFL=32760
ILRFONT  DCB   MACRF=R,DSORG=PO,RECFM=VBM,DDNAME=ILRFONT,             X
               EODAD=EPAKLOOP,BUFL=32760
ILRSYSPR DCB   MACRF=PM,DSORG=PS,RECFM=FBA,LRECL=121,BLKSIZE=5445,    X
               DDNAME=ILRSYSPR,BUFL=32760
ILRSYSIN DCB   MACRF=GM,DSORG=PS,RECFM=FB,LRECL=80,EODAD=EOFSYSIN,    X
               DDNAME=ILRSYSIN,BUFL=32760
ILRIN    DCB   MACRF=GL,DSORG=PS,DDNAME=ILRIN,EODAD=EOFPRTIN,         X
               BUFL=32760
ILROUT DCB     MACRF=PM,DSORG=PS,DDNAME=ILROUT,RECFM=VBA,             X
               BUFL=32760
         LTORG
*********************************************************************
*      DSECT TO MAP RESOURCE TABLE ENTRIES                          *
*********************************************************************
RESTBENT DSECT
RESTTYPL DS    CL8
RESTDCB  DS    F
         DS    AL3
RESTTYPC DS    AL1
*********************************************************************
*      DSECT TO MAP RESOURCE LIB INPUT BUFFER AREA                  *
*********************************************************************
RESBUF   DSECT
RESRDWP  DS    AL4
RESEOBP  DS    AL4
RESBDW   DS    F
RESRDW   DS    F
RESREC   DS    F
*********************************************************************
*      DSECT TO MAP WORKAREA FOR NESTLIST AND PACKLIST              *
*   PACKLIST AND NESTLIST ENTRIES HAVE THE SAME FORMAT (WHICH IS    *
*   ALSO THE SAME AS THE PACK REQUEST FORMAT IN THE SYSIN STREAM)   *
*   THE ENTRIES IN THESE LISTS ARE MAPPED BY THE WORKLIST DSECT     *
*********************************************************************
PACKWORK DSECT
PAKWRKID DS    D
PAKLSTRT DS    F
PAKLNEXT DS    F
PAKLMAX  DS    F
NSTLSTRT DS    F
NSTLCURR DS    F
```

```
NSTLNEXT DS    F
NSTLMAX  DS    F
LISTAREA EQU   *
*********************************************************************
*      DSECT TO MAP A WORKLIST ENTRY                                *
*********************************************************************
WORKLIST DSECT
WRKLNTRY DS    OCL17
WRKLTYPE DS    CL8
         DS    CL1
WRKLNAME DS    CL8
WRKLNTLN EQU   *-WRKLNTRY
*********************************************************************
*      DSECT TO MAP AFPDS RECORDS FOR CHK ROUTINES                  *
*********************************************************************
AFPDS    DSECT
AFPDSRDW DS    F
AFPDSCC  DS    XL1    |X'5A',RECLEN
AFPDSLEN DS    XL2    |RECORD LENGTH
AFPDSOP  DS    XL3    |AFPDS OPERATION CODE
AFPDSFLG DS    XL1    |AFPDS FLAG BYTE
AFPDSSEQ DS    XL2    |AFPDS RECORD SEQUENCE
AFPDSDAT EQU   *
MMOREC   EQU   *
MMORGLEN DS    CL1    |LENGTH OF MMO REPEATING GROUPS
         DS    CL3
MMORG    EQU   *      |START OF MMO REPEATING GROUPS (MAX 127)
MMORGOFF EQU   (*-AFPDSRDW)  |OFFSET ORIGIN FOR REPEAT GROUPS
         DS    CL4    |FLAGS ETC.
MMONAME  DS    CL8    |NAME OF OVERLAY
         ORG   AFPDSDAT
MPSREC   EQU   *
MPSRGLEN DS    CL1    |LENGTH OF MPS REPEATING GROUPS
         DS    CL3
MPSRG    EQU   *      |START OF MPS REPEATING GROUPS (MAX 127)
         DS    CL4    |FLAGS ETC.
MPSNAME  DS    CL8    |NAME OF PAGE SEGMENT
         ORG   AFPDSDAT
MCFREC   EQU   *
MCFRGOFF EQU   (*-AFPDSRDW)  |OFFSET ORIGIN FOR REPEAT GROUPS
MCFRGLEN DS    CL1    |LENGTH OF MCF REPEATING GROUPS
         DS    CL3
MCFRG    EQU   *      |START OF MCF REPEATING GROUPS
         DS    CL4    |FLAGS ETC.
MCFCFNAM DS    CL8    |NAME OF CODED FONT
MCFCPNAM DS    CL8    |NAME OF CODE PAGE
MCFCSNAM DS    CL8    |NAME OF CHARACTER SET
         ORG   AFPDSDAT
CFIREC   EQU   *
CFICSNAM DS    CL8    |CHARACTER SET NAME
CFICPNAM DS    CL8    |CODE PAGE NAME
         ORG   AFPDSDAT
IPSREC   EQU   *
IPSNAME  DS    CL8    |PAGE SEGMENT NAME
*********************************************************************
*      DSECT TO MAP BPAM DCBS                                       *
*********************************************************************
         DCBD  DSORG=(PO),DEVD=(DA)
         END   ILRPACK
```

A.6 LN2AFPDS Program

This utility program converts System/370 line-data print files (that is, 1403 or 3800-1 print files) into AFPDS. This is done by processing the file through an AFP pagedef in much the same fashion as PSF does.

The pagedef is an AFP object that provides a “mapping” from line-data to a finished composed page. Thus, it defines such specifications as line spacing and placement, the number of lines on a page, fonts used, and page orientation (portrait or landscape). After processing, all these variables are defined within the AFPDS output and the need for pagedef processing by PSF is removed. Therefore, the output AFPDS file is printable on systems that do not have the necessary pagedef, or that do not support pagedef processing at all.

Another advantage of converting a line-data file to AFPDS is that AFPDS is a self-defining datastream that is easy to transport. Some platforms do not use the concept of records, or use different code points for text, or do not understand variable length records. In all of these cases, transport of AFPDS is successful, whereas line-data may be clobbered.

A.6.1 Main PL/I Coding

```
*PROCESS MAR(2,72,1) AG A(F) LMSG INC M MAP NEST NSYN(S) STMT X(F);
*PROCESS NAME('LN2AFPN') NOCOUNT NOFLOW NGS OPT(TIME) CMP(V2);
 /* LAST UPDATE ON 13 FEB 1990 AT 10:07:47 BY VEND730 VERSION 01 */
 /* COPYRIGHT NOTICE                                                */
 %dcl debug char;                          /* debug-global switch     */
 %debug = 'no';                            /* debug-on = 'debug'      */
 LN2AFPN: PROCEDURE(RUN_TIME_PARM) OPTIONS(MAIN);
 /*****************************************************************/
 /*                                                               */
 /*                                                               */
 /*      A program to create an AFP data stream from line data    */
 /*      using a PAGEDEF to format the pages.                     */
 /*                                                               */
 /* This program takes line data input and a PAGEDEF input and does  */
 /*    the following:                                             */
 /*  1.Reads LND and associated structures into acquired memory.  */
 /*  2.Reads lines sequentially from line input, processing them  */
 /*    against the page definition to emit composed text          */
 /*    (presentation text) structured fields suitable for printing   */
 /*    by any PSF.                                                */
 /*                                                               */
 /*  THIS PROGRAM WHEN COMPILED WILL ISSUE A WARNING MESSAGE      */
 /*  IEL0872I  ADDR BUILTIN FUNCTION POINTS AT STRING LENGTH FIELD.  */
 /*                                                               */
 /*  FOR FURTHER INFORMATION, SEE THE AFP DATA STREAM REFERENCE   */
 /*  (S544-3202).                                                 */
 /*                                                               */
 /* written by: Howard L. Turetzky, Boulder Programming Center.   */
 /*             VEND605 at BLDVM2, (303)924-9079.                 */
 /*                                                               */
 /* COPYRIGHT 1989, 1990, IBM CORPORATION.                        */
 /*****************************************************************/
 /*****************************************************************/
 /*  Global data declarations.                                    */
 /*****************************************************************/

 DECLARE
   (ADDR, BIN, BIT, CHAR, DATE, INDEX, LENGTH, LOW, FIXED, MAX, MIN,
    HIGH, NULL, SUBSTR, TIME, TRANSLATE, UNSPEC)
               BUILTIN;                 /* DECLARE BUILT IN FCTNS.  */
```

```
DECLARE
   CPR        CHAR(30) STATIC INIT('COPYRIGHT 1989, 1990 IBM CORP'),
   VID        char(60) static
      init('H.L. Turetzky, IBM Boulder Programming Center, v.1.0'),
   DUMPLBL    CHAR(80) STATIC INIT('NOREASON'),
   DUMPOPTS   CHAR(10) STATIC INIT('TFSHB'),    /*PLIDUMP OPTIONS   */
   PGMNAME    CHAR(8)  STATIC INIT('LN2AFPDS'), /* PROGRAM NAME      */
   RUN_TIME_PARM   char(44) varying,    /* execution parm string    */
   (CC_Parm, TRC_Parm) char(4) var,       /* invocation parameters   */
   Chars_Parm          char(28) var,      /* for CHARS= parm         */
   (i,j)      fixed bin,                  /* misc indexes, counters  */
   AFPHdrLen      fixed bin(7) init(9),   /*cc+structured fld header */
   TRUE       BIT(1) static init('1'B) aligned,
                                         /* Flags for loop          */
   FALSE      BIT(1) static init('0'B) aligned;
                                          /* and file control.       */
  declare linedata_started bit(1) static;
dcl /* global paramter settings flags.                               */
   NOCC           bit(1),                /* uses no carriage control*/
   Mach_CC        bit(1),                /* machine carriage control*/
   ANSI_CC        bit(1),                /* ANS carriage control    */
   TRC            bit(1),                /* input has TRC byte      */
   CHARS          bit(1) init(false);    /* global CHARS valid flag */
/*********************************************************************/
/*  global dynamic structure pointers, list anchors, and list       */
/*  templates.                                                       */
/*********************************************************************/
DECLARE
 CCP_List_start              pointer        /*1st ccp for PAGEDEF  */
                             static init(null),
 prevptr                     pointer       /* previous list entry  */
                             static init(null);
DECLARE
 PDEF_List_Anchor            pointer        /* pagedef records list*/
                             static init(null),
 PDEFPTR                     pointer        /* pagedef record      */
                             static,
 PDEF_Reclen                 fixed bin,     /* input record length */
 1  PDEF_Rec                 based(PDEFPTR),        /*pagedef list */
    5  PDEF_Next             pointer,       /* next pagedef line   */
    5  PDEF_Length           fixed bin,     /* length of record    */
    5  PDEF_Data             char(PDEF_Reclen refer (PDEF_Length))
                                          /* variable length data*/;
DECLARE
/* There is one PFMT record for each page format in the page
   definition.  It contains information about the page's active
   environment in the form of pointers to the start of the
   active environment group and the first record and record count
   of objects that may occur more than once.  There is also a
   pointer to the LND list associated with this PAGEDEF.          */
 PFMT_List_Anchor            pointer      /*page format list head*/
                             static init(null),
 PFMTPTR                     pointer      /*page format entry    */
                             static,
 1  PFMT_Rec                 based(PFMTPTR),    /*page fmt list  */
    5  PFMT_Next             pointer,     /* next page format    */
    5  PFMT_Data,
       10  PFMT_start        pointer      /* start of this format*/
                             init(null),
       10  PFMT_name         char(8),     /* format token name   */
       10  PFMT_MCF,
           15  PFMT_1st_MCF  pointer,     /* 1st mcf this datamap*/
           15  PFMT_MCF_cnt  fixed bin(15)  /* number of mcf recs  */
                             init(0),
       10  PFMT_MPS,
           15  PFMT_1st_MPS  pointer,       /* 1st MPS this datamap*/
```

```
          15  PFMT_MPS_cnt   fixed bin(15)  /* number of MPS recs  */
                             init(0),
       10  PFMT_LND_List      pointer      /* LND list for this map */
                             init(null),
       10  PFMT_FDX,
           15  PFMT_1st_FDX   pointer,     /*   1st FDX this datamap*/
           15  PFMT_FDX_cnt   fixed bin(15)  /* number of FDX recs  */
                             init(0),
           15  PFMT_bytes     fixed bin(31); /* number of FDX bytes */
DECLARE
/* There is one LND record and array for each page format.  It
   contains all of the LND fields, in order, in the data map, in
   the same form as they appear in the LND structured field.
   The LND count is taken from the LNC record.                    */
 1  LND_List           based(PFMT_LND_List),      /*page fmt list   */
    5  LNDLST_LNC      fixed bin(15),     /*   line descriptor count*/
    5  LNDLST_array    (LNC_FDS_cnt refer (LNDLST_LNC)),
       10  LNDLST_data  unaligned,        /* contents of an LND      */
         15  LND_flags      char(2),      /* flag bits               */
         15  LND_inline     fixed bin(15),/* inline position         */
         15  LND_baseln     fixed bin(15),/* baseline position       */
         15  LND_orient     char(4),      /* text orientation        */
         15  LND_fontid     char(1),      /* local font identifier   */
         15  LND_channel    char(1),      /* channel code            */
         15  LND_nxt_skip   fixed bin(15),/* next LND if skipping    */
         15  LND_nxt_spc    fixed bin(15),/* next LND if spacing     */
         15  LND_nxt_reuse  fixed bin(15),/* next LND if reusing data*/
         15  LND_supp_token char(8),      /* suppression token name  */
         15  LND_reserved   char(1),
         15  LND_data_start fixed bin(31),/* data start position     */
         15  LND_data_len   fixed bin(15),/* data length             */
         15  LND_color      char(2),      /* text color value        */
         15  LND_nxt_cond   fixed bin(15),/* next LND if cond. proc. */
         15  LND_subpage    char(1),      /* subpage id              */
         15  LND_ccp_id     fixed bin(15),/* CCP identifier          */
 1  LND_List_rdef      based,                    /*redefinition    */
    5  LNDLST_LNC_rdef  fixed bin(15),     /*   line descriptor count*/
    5  LNDLST_data_rdef (LNC_FDS_cnt refer (LNDLST_LNC_rdef))
                        char(40);         /* contents of an LND      */
declare                                   /* LND flag values         */
   lndflag_endskip   bit(16) static        /*end page if skipping    */
                     init('1000000000000000'b),
   lndflag_endspc    bit(16) static        /*end page if spacing     */
                     init('0100000000000000'b),
   lndflag_inline    bit(16) static        /*generate inline position*/
                     init('0010000000000000'b),
   lndflag_baseline  bit(16) static        /*generate baseline posn. */
                     init('0001000000000000'b),
   lndflag_fontchg   bit(16) static        /*generate font change    */
                     init('0000100000000000'b),
   lndflag_suppress  bit(16) static        /*generate suppression    */
                     init('0000010000000000'b),
   lndflag_reuse     bit(16) static        /*reuse record            */
                     init('0000001000000000'b),
   lndflag_fixdata   bit(16) static        /*used fixed data         */
                     init('0000000100000000'b),
   lndflag_TRC       bit(16) static        /*use compatibility trc   */
                     init('0000000001000000'b),
   lndflag_color     bit(16) static        /*set text color          */
                     init('0000000000100000'b),
   lndflag_condproc  bit(16) static        /*conditional processing  */
                     init('0000000000010000'b);
/*****************************************************************/
/*  FILE DECLARES-NOTE SYSPRINT IS PL/I DEFAULT FOR OUTPUT DEVICE   */
/*****************************************************************/
DECLARE
```

```
   Pdeflib      FILE RECORD INPUT,      /* page definition file     */
   Pagedef_EOF BIT(1) STATIC INIT('0'B), /* END OF FILE FLAG        */
   SYSPRINT FILE STREAM OUTPUT ENV (F RECSIZE(080)); /* message file*/
/*********************************************************************/
/* AFPDS record descriptions: each type of record referenced is      */
/*   defined based on a pointer.  Records read from the PAGEDEF are */
/*   referenced with the same pointer (INREC_PTR).  Records to be    */
/*   written are referenced with OUTREC_PTR.  Some record types      */
/*   (eg., in the active environment) may be both read and written. */
/*********************************************************************/
DECLARE
 1 AFPDSREC BASED(InRec_ptr) UNALIGNED, /* general afpds record     */
   5  CC         CHAR(1) ,            /* Carriage control 5A  hex   */
   5  COUNT      FIXED BIN(15),       /* Length AFPDS record        */
   5  TYPE       bit(24),             /* Type of AFPDS record       */
   5  FLAG       bit(8),                  /* flag byte               */
   5  SEQUENCE   fixed bin(15),           /* structure sequence num. */
   5  REST       CHAR(32758)          /* Rest of the record         */;
DECLARE
 InRec_ptr       pointer static,          /* input record pointer    */
 AFPDString based(InRec_ptr) unaligned    /* string for assignment   */
                 char(32767);
DECLARE
 1 CNTREC  BASED(InRec_ptr) UNALIGNED,    /* count records(LNC,FDS)  */
   5  CC         CHAR(1) ,                /* Carriage control 5A hex */
   5  COUNT      FIXED BIN(15),           /* Length AFPDS record     */
   5  TYPE       bit(24),                 /* Type of AFPDS record    */
   5  FLAG       bit(8),                  /* flag byte               */
   5  SEQUENCE   fixed bin(15),           /* structure sequence num. */
   5  LNC_FDS_cnt fixed bin(15);          /* number of lnd records   */
DECLARE
 1 LNDREC  BASED(InRec_ptr) UNALIGNED,    /* line descriptor record  */
   5  CC             CHAR(1) ,            /* Carriage control 5A hex */
   5  COUNT          FIXED BIN(15),       /* Length AFPDS record     */
   5  TYPE           bit(24),             /* Type of AFPDS record    */
   5  FLAG       bit(8),                  /* flag byte               */
   5  SEQUENCE   fixed bin(15),           /* structure sequence num. */
   5  LNDREC_data char(40);               /* LND structured fields   */
DECLARE
/* The MSU record occurs in a FORMDEF, and relates text suppression
   identifiers (named in LNDs) to suppression identifiers (used in
   BSU/ESU text controls).  If suppression is used, a FORMDEF (or at
   least an MSU record) must occur in the PDEFLIB input or all text
   suppression will be ignored.                                     */
    MSU_cnt             fixed bin(15),    /* repeating groups       */
    (MSUPTR,MSUdptr)    pointer           /* to the MSU, or null    */
                        init(NULL),
 1  MSU_Record          based(MSUPTR),    /*  map suppression record */
    5  MSU_Num_Gps      fixed bin(15),    /* number of repeating grps*/
    5  MSU_array        (MSU_cnt refer (MSU_Num_Gps)),
       10  MSU_data     unaligned,        /* contents of the MSU    */
         15  MSU_supp_token char(8),      /* suppression token name */
         15  MSU_reserved   char(1),
         15  MSU_supp_id    char(1),      /* suppression identifier */
   MSU_data_rdef based unaligned          /* string definer for array*/
                        char(1280);
DCL
 /*******************************************************************/
 /* The MCF is mapped to allow calculation of the high water mark   */
 /* font index used for TRC error detection, and for constructing   */
 /* an MCF to be used if CHARS= is specified.                       */
 /*******************************************************************/
    (TRC_Char_Font_cnt,                     /*compat (TRC) max fonts*/
     TRC_Font_cnt,                        /*non-compat max fonts    */
     MCF_cnt)    fixed bin(15) init(0),   /* calculate number of gps */
    MCF_REC_Ptr              pointer,    /* to MCF records          */
```

```
   MCF_ent_len  fixed bin(8) init(30),   /* entry len for fake MCF  */
 1  MCF_REC      BASED(MCF_REC_Ptr) unaligned,
    5  MCF_REC_len          fixed bin,    /*    length prefix for wrt*/
    5  MCF_record           unaligned,    /* actual record           */
       10 CC                CHAR(1),      /* Carriage control 5A hex */
       10 COUNT             FIXED BIN(15),/* Length AFPDS record     */
       10 TYPE              bit(24),      /* Type of AFPDS record    */
       10 FLAG              bit(8),       /* flag byte               */
       10 SEQUENCE          fixed bin(15),/* structure sequence num. */
       10 MCF_GRP_LEN       bit(8),       /* repeating element len   */
       10 FILLER            CHAR(1),
       10 MCF_rpt_grp       fixed bin(15),   /* number of elements   */
          /*note: mcf_rpt_grp is not a field in the MCF. it is       */
          /* unchecked filler used here as a place to keep the       */
          /* array counter required by PL/I.                         */
       10 MCF_GRP (MCF_cnt refer (MCF_rpt_grp)), /*font mapping array*/
           15 MCF_CFLI        char(1),    /*   coded font local id   */
           15 FILLER          CHAR(1),
           15 MCF_CFSI        char(1),    /* double-byte section id  */
           15 FILLER2         CHAR(1),
           15 (MCF_CODED_FONT,            /* font name               */
                MCF_CODE_PAGE,            /* code page name          */
                MCF_CHAR_SET)  CHAR(8),   /* character set name      */
           15 MCF_ROTATION     CHAR(2);   /* rotation value          */
DECLARE
 1 CCPList BASED(CCPPTR) UNALIGNED, /* ccp record list entry        */
   5  Next_CCP_ptr      pointer,      /* next record in list        */
   5  CCP_Rec,
     10 CCP_Id          fixed bin(15), /* ccp identifier             */
     10 Next_CCP        fixed bin(15), /* next ccp to process        */
     10 Subpage_Flag    char(1),       /* subpage action flags       */
     10 reserved        char(1),
     10 Rpt_Grp_Cnt     fixed bin(15), /* number of repeating groups */
     10 Rpt_Grp_Len     fixed bin(15), /* length of repeating group  */
     10 Compare_Len     fixed bin(15), /* length of comparison string*/
     10 CCP_Group                      /* ccp repeating group        */
                        (ccprec.rpt_grp_cnt refer
                         (CCPList.rpt_grp_cnt)),
        15  Timing       char(1),      /* timing of action           */
                                      /* 0=take default action
                                         1=take action immediately
                                         2=before current subpage
                                         -127=after current line
                                         -126=after current subpage */
        15  Med_Map_Act  char(1),      /* medium map action          */
                                      /* 0=ignore
                                         1=use current with pg eject
                                         2=invoke named medium map
                                         3=invoke first medium map
                                         4=invoke next medium map    */
        15  Med_Map_Nm   char(8),      /* medium map name            */
        15  Data_Map_act char(1),      /*data   map action          */
                                      /* 0=ignore
                                         1=use current with pg eject
                                         2=invoke named data map
                                         3=invoke first data map
                                         4=invoke next data map      */
        15  Data_Map_Nm  char(8),      /* medium map name            */
        15  Comparison   char(1),      /* compare type               */
                                      /* 0=any change
                                         1=equal to
                                         2=less than
                                         3=equal to or less than
                                         4=greater than
                                         5=equal to or greater than
                                         6=not equal
```

```
                                        7=take action w/o compare */
       15  Compare_string           /* comparison string         */
                   char(ccprec.Compare_len refer
                        (CCPList.Compare_len));
 %PAGE;
 %DCL GEN CHAR;
 %GEN = 'BIT';
 %include sfidequ;                      /* structured field defs    */
 /*=================================================================*/
 /*+++++++++>>>>>>>>>       BEGIN MAINLINE        <<<<<<<<<+++++++++*/
 /*=================================================================*/
 on error begin;
  DUMPLBL = 'ON ERROR EXIT  ';
  call PLIDUMP(dumpopts,dumplbl);
 end;
 on condition(substringrange);
 on condition(stringrange);
 call Process_Parms;                    /* set up invocation parms */
 call parse_pagedef;                    /* prepare pagedef         */
%if debug = 'debug' %then %do;          /* debug                   */
 pfmtptr = pfmt_list_anchor;            /* debug                   */
 do until (pfmtptr   = null);           /* debug                   */
   put edit('PFMT name: ',pfmt_name)    /* debug                   */
       (skip,a,a);                      /* debug                   */
   do i=1 to lndlst_lnc;                /* debug                   */
    (noconversion):                     /* debug                   */
     put edit(i,                        /* debug                   */
              unspec(lnd_flags(i)),     /* debug                   */
              lnd_inline(i),            /* debug                   */
              lnd_baseln(i),            /* debug                   */
              unspec(lnd_fontid(1)),    /* debug                   */
              unspec(lnd_channel(1)),   /* debug                   */
              lnd_nxt_skip(i),          /* debug                   */
              lnd_nxt_spc(i),           /* debug                   */
              lnd_nxt_reuse(i))         /* debug                   */
        (skip,f(4),x(1),b(12),2(x(1),f(4)), /* debug               */
             2(x(1),b(8)),3(x(1),f(4)));  /* debug                 */
   end;                                 /* debug                   */
   pfmtptr = pfmt_next;                 /* debug                   */
 end;                                   /* debug                   */
%end;                                   /* debug                   */
 call Process_Linedata;                 /* create AFPDS output     */
%if debug = 'debug' %then %do;          /* debug                   */
 dumplbl = 'PAGEDEF Processing';        /* debug                   */
 call plidump(dumpopts,dumplbl);        /* debug */
%end;                                   /* debug                   */
Process_Parms: proc;
 /* Process run-time parms.  These parms may be determined by flags  */
 /* from the caller or from JCL using a different API.               */
 /* determine carriage control type from parameter or external rtn.  */
dcl
 fontstr (4)      char(8) var,          /* font name array         */
 null_font        char(8) unaligned     /* value for null MCF font */
                  init((high(2)||low(6)));
 if length(Run_Time_Parm) > 0           /* was execution parm      */
  then do;
   CC_Parm = substr(Run_Time_Parm,1,4);   /* carriage control type   */
   TRC_Parm = substr(Run_Time_Parm,6,3);  /* using TRC?              */
   ANSI_CC = (CC_Parm = 'ANSI');          /* determine carriage      */
   Mach_CC = (CC_Parm = 'MACH');          /*control type from parm.  */
   NOCC  = ¬(ANSI_CC | Mach_CC);          /* set by parm or ext. rtn.*/
   TRC   = (TRC_Parm = 'TRC');            /* set by parm or ext. rtn.*/
   Chars_Parm = substr(Run_Time_Parm,10); /* look for CHARS=(font...)*/
   /* validate parms and report them.                               */
   if length(Chars_Parm) > 0            /* validate user fonts     */
    then
```

```
     if substr(Chars_Parm,1,7) = 'CHARS=('
      then do;
       i = 8;                              /* start of font names     */
       do until (i >= length(Chars_Parm));  /* parse the list          */
        j = index(substr(Chars_Parm,i),',');  /* list separator        */
        if j = 0                           /* no comma--may be last   */
         then
          j = index(substr(Chars_Parm,i),')');  /* list end           */
        if j = 0
         then
          call ErrorExit(10,Chars_Parm,' ',' '); /* terminate w/msg */
        TRC_Char_Font_cnt = TRC_Char_Font_cnt + 1; /* maximum fonts  */
        fontstr(TRC_Char_Font_cnt) = 'X0' ||
                           substr(Chars_Parm,i,(j-1)); /*coded font*/
        i = j + i;                         /* next font in list       */
       end/*until i>=parm*/;
  /* create and initialize the MCF for use with TRC and CHARS=       */
       MCF_cnt = TRC_Char_Font_cnt;        /* alloc size for TRC MCF  */
       allocate MCF_REC;                   /* make a TRC font map     */
       MCF_REC.cc = '!';                   /* make it like a real MCF */
       MCF_REC.count = MCF_cnt * MCF_ent_len + 12; /* AFP rec length */
       MCF_REC_len = MCF_REC.count + 1;    /* record len + cc to write*/
       MCF_REC.type = SF_MCF;              /* identify as MCF record  */
       MCF_REC.flag = '00000000'b;
       MCF_REC.sequence = 0;
       MCF_GRP_len = substr(unspec(MCF_ent_len),9,8);
                                           /* use max group entry len */
       do i = 1 to TRC_Char_Font_cnt;      /* set up each entry       */
        unspec(MCF_CFLI(i)) = substr(unspec(i),9,8);
                                           /*     local font id       */
        MCF_CFSI(i) = low(1);              /* for double-byte fonts   */
        MCF_CODED_FONT(i) = fontstr(i);    /* fill in font name       */
        MCF_CODE_PAGE(i) = null_font;           /* null for code pg  */
        MCF_CHAR_SET(i)  = null_font;           /* null for charset  */
        MCF_ROTATION(i)  = low(2);         /* 0-degree rotation       */
       end/*1 to font count*/;
       CHARS = true;                       /* set global CHARS flag   */
      end/*if CHARS=*/;
      else                                 /* invalid CHARS parm      */
       call ErrorExit(10,Chars_Parm,' ',' '); /* terminate w/msg     */
   end/*run_time_parm>0*/;
  else;                                    /* call external routine   */
end Process_Parms;
Parse_PAGEDEF: proc;
/*********************************************************************/
/*** Build the list structures needed for processing line data.    ***/
/*** each page format is identified and its environment and lnd    ***/
/*** records indexed.                                              ***/
/*********************************************************************/
/*==> Any construct that creates an array giving data start,       ***/
/*** orientation, font, next line, and other LND parameters could  ***/
/*** be used in place of a PAGEDEF.                                ***/
/*********************************************************************/
/*********************************************************************/
/*** Reads PAGEDEF records and returns a pointer to them.  Null is ***/
/***returned at EOF.                                               ***/
/*********************************************************************/
 Get_PDEF: proc returns(pointer);
 dcl
   recptr         pointer;
   read file(pdeflib) set(recptr);      /* get pointer to next rec   */
   if Pagedef_EOF                       /* eof?                      */
    then
     return(null);                      /* return eof indication     */
    else
     return(recptr);                    /* return record pointer     */
```

```
 end Get_PDEF;
/*********************************************************************/
/*  Saves pagedef records into a list in memory.                     */
/*********************************************************************/
 Store_PDEF: procedure(recptr) returns(pointer);
 dcl
   recptr         pointer;               /* record pointer parameter*/
 pdef_reclen = recptr->AFPDSREC.count + 1; /* afpds record length    */
 if PDEF_List_anchor = null             /* 1st record in list?     */
  then do;                              /* initialize list         */
   allocate PDEF_Rec;                   /* first record            */
   prevptr = PDEFPtr;                   /* save for forward chain  */
   PDEF_List_anchor = PDEFPtr;          /* anchor the list head    */
   PDEF_Next = null;                    /* no next entry yet       */
   PDEF_Data = substr(recptr->AFPDString, /* save data record of    */
                      1,pdef_reclen);   /* input data length       */
   return(PDEFPtr);                     /* give back list entry    */
 end;
 else do;                               /* add successor list items*/
  allocate PDEF_Rec set(prevptr->PDEF_Next); /* chain to last rec   */
  prevptr = prevptr->PDEF_Next;         /* save new entry pointer  */
  prevptr->PDEF_Next = null;            /* no next record yet      */
  prevptr->PDEF_Data =
              substr(recptr->AFPDString, /* save data record of    */
                     1,pdef_reclen);    /* input data length       */
  return(prevptr);                      /* give back list entry    */
 end;
end Store_PDEF;
/*************************************************************/
/*** Read each PAGEDEF record and store or bypass as        ***/
/*** needed.  Framing and constant records are ignored,     ***/
/*** but checked for proper begin-end structure.            ***/
/*************************************************************/
dcl
 (found_PGD, found_CTC, found_PTD,      /* required record flags   */
  was_BPM, was_BAG, was_BDX) bit(1) init(false),
 (PFMT_cnt, edm_cnt, ccp_cnt)
                      fixed bin init(0), /* structure counters     */
 LND_indx                   fixed bin,  /* LND array subscript     */
 (PFMT_prev_ptr, tempptr)   pointer init(null); /* list temporaries*/
OPEN FILE (Pdeflib);
ON ENDFILE (Pdeflib) Pagedef_EOF = true;
InRec_Ptr = Get_PDEF;                   /* page def priming read   */
do until (PAGEDEF_EOF);                 /* process all PDEF records*/
  select (AFPDSREC.Type);
   when(SF_BPM) do;                     /* must be only one        */
    if was_BPM
     then
      call ErrorExit(1,' ',' ',' ');    /* report error and exit   */
     else
      was_BPM = true;
   end /*SF_BPM*/;
   when(SF_EPM) do;                     /* must be only one        */
    if was_BPM
     then
      ;
     else
      call ErrorExit(2,' ',' ',' ');    /* report error and exit   */
   end /*SF_EPM*/;
   when(SF_BDM) do;
    PFMT_cnt = PFMT_cnt + 1;            /* count data maps         */
    if PFMT_cnt = 1
     then do;
      alloc PFMT_REC;                   /* data map list header    */
      PFMT_prev_ptr = PFMTptr;          /* save new rec pointer    */
      PFMT_List_Anchor = PFMTptr;       /* page map list head      */
```

```
  end;
  else do;                              /* next data map list entry*/
   alloc PFMT_REC set(PFMT_prev_ptr->PFMT_next);
   PFMT_prev_ptr = PFMT_prev_ptr->PFMT_next;   /*forward chain   */
   PFMT_prev_ptr->PFMT_next = null;   /* list end               */
   PFMTptr = PFMT_prev_ptr;           /* for reference this PFMT */
  end;
 PFMT_name = substr(AFPDSREC.rest,1,8);/* identifies map name    */
 PFMT_start = STORE_PDEF(InRec_ptr);  /* start of this data map  */
end /*SF_BDM*/;
when(SF_EDM) do;                       /* must be only one        */
 edm_cnt = edm_cnt + 1;                /* end map count           */
 if edm_cnt ¬= PFMT_cnt                /* same begin/end counts   */
  then
   call ErrorExit(5,'EDM','BDM',' '); /* quit if unmatched       */
 end /*SF_EDM)*/;
when(SF_BAG) do;                       /* must be only one        */
 if was_BAG
  then
   call ErrorExit(3,' ',' ',' ');     /* report error and exit   */
  else
   was_BAG = true;                     /* note active env. start  */
end /*SF_BAG*/;
when(SF_EAG) do;                       /* must be only one        */
 if was_BAG
  then
   was_BAG = false;                    /* reset for next environ. */
  else
   call ErrorExit(4,' ',' ',' ');     /* report error and exit   */
end /*SF_EAG*/;
when(SF_BDX) do;                       /* datamap subcase match   */
 if was_BDX
  then
   call ErrorExit(6,'BDX','EDX',' '); /* report error and exit   */
  else
   was_BDX = true;                     /* note datamap subcase    */
end /*SF_BDX*/;
when(SF_EDX) do;                       /* datamap subcase match   */
 if was_BDX
  then
   was_BDX = false;                    /* reset for next subcase  */
  else
   call ErrorExit(7,'BDX', ' ',' ');  /* report error and exit   */
end /*SF_EDX*/;
when(SF_CCP) do;                       /* condition control       */
 ccp_cnt = ccp_cnt + 1;                /* number of CCPs          */
 tempptr   = STORE_PDEF(InRec_ptr);    /* save in PDEF list       */
 if ccp_cnt = 1                        /* first CCP               */
  then
   CCP_list_start = tempptr;           /*      start of CCPs      */
end /*SF_CCP*/;
when(SF_MCF) do;                       /* font list records       */
 PFMT_MCF_cnt = PFMT_MCF_cnt + 1;      /* number of MCFs          */
 tempptr   = STORE_PDEF(InRec_ptr);    /* save in PDEF list       */
 if PFMT_MCF_cnt = 1                   /* first MCF               */
  then
   PFMT_1st_MCF = tempptr;             /* start of MCFs for map   */
 tempptr = addr(tempptr->PDEF_Length); /* point to record prefix */
 TRC_Font_cnt = ((tempptr->MCF_Rec.count - AFPhdrlen - 1) /
                 unspec(tempptr->MCF_rec.MCF_GRP_len)) +
                TRC_Font_cnt;          /* max number of fonts     */
end /*SF_MCF*/;
when(SF_MPS) do;                       /* font list records       */
 PFMT_MPS_cnt = PFMT_MPS_cnt + 1;      /* number of MPSs          */
 tempptr   = STORE_PDEF(InRec_ptr);    /* save in PDEF list       */
 if PFMT_MPS_cnt = 1                   /* first MPS               */
```

```
      then
       PFMT_1st_MPS = tempptr;            /*    start of MPSs        */
     end /*SF_MPS*/;
     when(SF_LNC) do;                     /* line descriptor count   */
      if LNC_FDS_cnt > 0                  /* valid LND count?        */
       then do;
        alloc LND_List;                   /* LND array for this PFMT */
        LND_indx = 1;                     /* reset LND subscript     */
       end;
       else
        call ErrorExit(8,char(CNTREC.sequence),' ',' ');
                                          /* stop on invalid count   */
     end /*SF_LNC*/;
     when(SF_LND) do;                     /* line descriptors        */
      tempptr = PFMT_LND_list;            /* point to lnd array      */
      tempptr->LNDLST_data_rdef(LND_indx) =
                          LNDREC_data;    /* move entire LND         */
      LND_indx = LND_indx + 1;            /* number of next LND      */
     end /*SF_LND*/;
     when(SF_FDS) do;                     /* fixed data length       */
      PFMT_Bytes = LNC_FDS_cnt;           /* bytes of fixed text     */
     end /*SF_FDS*/;
     when(SF_FDX) do;                     /* fixed data text         */
      PFMT_FDX_cnt = PFMT_FDX_cnt + 1;    /* number of FDXs          */
      tempptr   = STORE_PDEF(InRec_ptr);  /* save in PDEF list       */
      if PFMT_FDX_cnt = 1                 /* first FDX               */
       then
        PFMT_1st_FDX = tempptr;           /*    start of FDXs        */
     end /*SF_FDX*/;
     when(SF_PGD) do;                     /* page descriptor (reqd)  */
      tempptr   = STORE_PDEF(InRec_ptr);  /* save in PDEF list       */
      found_PGD = true;                   /* mark found              */
     end /*SF_PGD*/;
     when(SF_PTD) do;                     /* pres txt descr. (reqd)  */
      tempptr   = STORE_PDEF(InRec_ptr);  /* save in PDEF list       */
      found_PTD = true;                   /* mark found              */
     end /*SF_PTD*/;
     when(SF_CTC) do;                     /* comp txt ctl (compat)   */
      tempptr   = STORE_PDEF(InRec_ptr);  /* save in PDEF list       */
      found_CTC = true;                   /* mark found              */
     end /*SF_CTC*/;
     when(SF_MSU) do;                     /* suppression map--FORMDEF*/
      MSU_cnt = (AFPDSrec.count - 8) / 10; /* number of repeating grps*/
      allocate MSU_Record;                /* save in it's own record */
      MSUdptr = addr(MSU_Array);          /* fill entire array       */
      substr(MSUdptr->MSU_data_rdef,1,(AFPDSREC.Count - 8)) =
                         substr(AFPDString,10,       /*start after hdr*/
                         (AFPDSRec.Count - 8)); /* length - header    */
     end /*SF_MSU*/;
     otherwise do;                        /* save all other recs     */
      tempptr   = STORE_PDEF(InRec_ptr);  /* save in PDEF list       */
     end;
    end /*select case of record type*/;
    InRec_Ptr = Get_PDEF;                 /* page def priming read   */
   end /*do until eof pagedef*/;
  close FILE (Pdeflib);
  if ¬found_PGD                           /* report missing records  */
   then
    call ErrorExit(5, 'EAG','PGD',' ');   /* terminate with message  */
  if ¬found_CTC                           /* report missing records  */
   then
    call ErrorExit(5, 'EAG','CTC',' ');   /* terminate with message  */
  if ¬found_PTD                           /* report missing records  */
   then
    call ErrorExit(5, 'EAG','PTD',' ');   /* terminate with message  */
 end Parse_PAGEDEF;
```

```
Process_LineData: PROC;
/*********************************************************************/
/* Line data records are read and processed against the appropriate  */
/* page format as determined by the carriage control byte and type.  */
/*                                                                   */
/*********************************************************************/
 dcl
/*********************************************************************/
/* constants                                                         */
/*********************************************************************/
   Mach_skip      bit(8) init('10000000'b)  /*mach carr ctl skip flag*/
                  aligned,                  /*                       */
   Mach_immed     bit(8) init('00000010'b)  /*mach carr ctl skip flag*/
                  aligned,                  /*                       */
   Mach_value     fixed bin(7) init(8),   /*mach skip/space amt shift*/
   ctl_5A         bit(8) init('01011010'b)     /* x'5A'              */
                                  aligned, /* structured fld cc  */
   ANSI_skc1      bit(8) init('11110001'b)     /* x'F1'              */
                                  aligned, /* skip channel 1     */
   ANSI_skc2      bit(8) init('11110010'b)     /* x'F2'              */
                                  aligned, /* skip channel 1     */
   ANSI_skc3      bit(8) init('11110011'b)     /* x'F3'              */
                                  aligned, /* skip channel 1     */
   ANSI_skc4      bit(8) init('11110100'b)     /* x'F4'              */
                                  aligned, /* skip channel 1     */
   ANSI_skc5      bit(8) init('11110101'b)     /* x'F5'              */
                                  aligned, /* skip channel 1     */
   ANSI_skc6      bit(8) init('11110110'b)     /* x'F6'              */
                                  aligned, /* skip channel 1     */
   ANSI_skc7      bit(8) init('11110111'b)     /* x'F7'              */
                                  aligned, /* skip channel 1     */
   ANSI_skc8      bit(8) init('11111000'b)     /* x'F8'              */
                                  aligned, /* skip channel 1     */
   ANSI_skc9      bit(8) init('11111001'b)     /* x'F9'              */
                                  aligned, /* skip channel 1     */
   ANSI_skcA      bit(8) init('11000001'b)     /* x'C1'              */
                                  aligned, /* skip channel 10    */
   ANSI_skcB      bit(8) init('11000010'b)     /* x'C2'              */
                                  aligned, /* skip channel 11    */
   ANSI_skcC      bit(8) init('11000011'b)     /* x'C3'              */
                                  aligned, /* skip channel 12    */
   ANSI_spc0      bit(8) init('01001110'b)     /* x'4E'              */
                                  aligned, /* print no space     */
   ANSI_spc1      bit(8) init('01000000'b)     /* x'40'              */
                                  aligned, /* print 1  space     */
   ANSI_spc2      bit(8) init('11110000'b)     /* x'F0'              */
                                  aligned, /* print 2  space     */
   ANSI_spc3      bit(8) init('01100000'b)     /* x'60'              */
                                  aligned, /* print 3  space     */
   page_SKP       bit(8) init('00000001'b)
                                  aligned, /* new page skip/space*/
   page_IDM       bit(8) init('00000010'b)
                                  aligned, /* new pageformat     */
   page_IMM       bit(8) init('00000011'b)
                                  aligned, /* new copygroup      */
   page_text      bit(8) init('00000100'b)
                                  aligned, /* text write force pg*/
   font_ids       char(10) unaligned,           /*for translate    */
   font_ids_tbl   (10) char(1) based      /* font id table x'00'-'09'*/
                  unaligned,
/*********************************************************************/
/* flags and counters global to line data processing                 */
/*********************************************************************/
   (End_Page_Skip,                        /* end page if skipping    */
    End_Page_Spc,                         /* end page if spacing     */
    New_Page,                             /* IDM new page created    */
```

```
    Pg_Pend,                              /* TOF skip pending        */
    held_IMM,                             /* pending IMM flag        */
    BPT_Open)     bit(1) aligned,         /* present. pg started     */
   chnl_code      bit(8) aligned,         /* channel control code    */
   ChNum          bit(8) aligned,         /* skip channel number     */
   CurrLND        fixed bin(15),          /* LND to use for printing */
   Page_cnt       fixed dec(07),          /* output page counter     */
   RecNum         fixed bin(15) init(0),  /* line input record cntr  */
   (LinePtr,                              /* to current line record  */
    OutRec_ptr)   pointer,                /* to output buffer        */
   ChCode         char(1),                /* channel code as char    */
   DataMap        char(8),                /* page format name        */
   Last_IMM       char(17) var,           /* last IMM record         */
   space_cnt      fixed bin(7);           /* line spacing counter    */
  Dcl
   LineIn       FILE RECORD INPUT,        /* Input line data file    */
   LineIn_EOF BIT(1) STATIC INIT('0'B),   /* END OF FILE FLAG        */
   1 LineRec       based      UNALIGNED,  /* line data input record  */
     5  len        fixed bin(15),         /* input record length     */
     5  CC         bit(8),                /* carriage control byte   */
     5  TRC_byte   char(1),               /* table reference (font)  */
   LineData       char(16384) varying,  /* line input data         */
   Line_len       fixed bin(15),         /* length of line data     */
   1 LineAFP BASED(InRec_ptr) UNALIGNED, /*general afpds in linedata */
     5  len        fixed bin(15),         /* input record length     */
     5  CC         CHAR(1) ,              /* Carriage control 5A hex */
     5  COUNT      FIXED BIN(15),         /* Length AFP data         */
     5  TYPE       bit(24),               /* Type of AFP record      */
     5  FLAG       bit(8),                /* flag byte               */
     5  SEQUENCE   fixed bin(15),         /* structure sequence num. */
     5  REST       CHAR(32756),           /* Rest of the record      */
   AFPDS        FILE RECORD output,       /* AFPDS output            */
   1 AFPout BASED(OutRec_ptr) UNALIGNED,  /*general afpds output     */
     5  len        fixed bin(15),         /* input record length     */
     5  CC         bit(8) aligned,        /* Carriage control 5A hex */
     5  COUNT      FIXED BIN(15),         /* Length AFP data         */
     5  TYPE       bit(24),               /* Type of AFP record      */
     5  FLAG       bit(8),                /* flag byte               */
     5  SEQUENCE   fixed bin(15),         /* structure sequence num. */
     5  Data       CHAR(16380);           /* 8K records              */
  Get_Line: proc returns(pointer);
/*********************************************************************/
/* Read a line data input line and return a pointer to it            */
/*********************************************************************/
  dcl
   recptr         pointer;
   read file(LineIn) into (LineData);     /* get pointer to next rec */
   line_len = length(LineData);           /* set global to data len  */
   if LineIn_EOF                          /* eof?                    */
    then
     return(null);                        /* return eof indication   */
    else do;
     RecNum = RecNum + 1;                 /* count the input record  */
     return(addr(LineData));              /* return record pointer,  */
                                          /* point past length       */
    end;
  end Get_Line;
  Write_AFP: Proc(DataPtr,LNDnum,DontPrnt) recursive;
/*********************************************************************/
/* Constructs AFP data from line data using LNDnum LND specification.*/
/* AFP records that have been input or constructed are written       */
/* directly (LNDnum=0), otherwise Present_Text is called to create   */
/* a presentation text record from input data and LNDnum.            */
/*********************************************************************/
  dcl
   DataPtr             pointer,           /* to line data            */
```

```
   AFPstring           char(32767) var based, /* string overlay      */
   DontPrnt            bit(1) aligned,    /* postion without printing*/
   LNDnum              fixed bin(15);     /* LND index              */
   if LNDnum ¬= 0                         /* format into PTX record  */
    then do;
     if Pg_Pend                           /* hanging new page?       */
      then
       call Page_Environ(Page_cnt,page_text); /* new page before text*/
     call Present_Text(LNDnum,LineData,DontPrnt); /*format text      */
    end;
   write file(AFPDS) from(DataPtr->AFPstring);   /*length+AFP record */
  end Write_AFP;
  Find_Skip: PROC(CC,StartLND) returns(fixed bin);
/*********************************************************************/
/* Finds LND to use for skip to channel command. If no channel is    */
/* found, a message is printed and single spacing is used.           */
/* Because ANSI control is control before write, a StartLND value of */
/* zero is used to signal beginning the channel search at LND 1      */
/* instead of the LND after the first one.                           */
/*********************************************************************/
  dcl
   CC             bit(8) aligned,         /* channel code char value */
   Skip_Top       bit(1),                 /* save skip form flag     */
   (StartLND, NextLND, FirstSkpLND)
                  fixed bin(15);          /* LND array indexes       */
   if StartLND = 0                        /* initial case ANSI       */
    then
     NextLND = 1;                         /* start at first LND      */
    else
     NextLND = LND_Nxt_skip(StartLND);    /* start with next skip    */
   FirstSkpLND = NextLND;                 /* save starting point     */
   call Set_EndPage_Flags(NextLND);       /* was this a TOF?         */
   if End_Page_Skip                       /* save page eject flag    */
    then
     Skip_Top = true;                     /* must skip to next page  */
    else
     Skip_Top = false;                    /* no eject on 1st try     */
   do while (unspec(LND_Channel(NextLND)) ¬= CC); /*find chnl code   */
    NextLND = LND_Nxt_skip(NextLND);      /* next LND in skip chain  */
    call Set_EndPage_Flags(NextLND);      /* was this a TOF?         */
    if End_Page_Skip                      /* save page eject flag    */
     then
      Skip_Top = true;                    /* must skip to next page  */
    if (NextLND = FirstSkpLND) &          /* back to start and       */
       (unspec(LND_Channel(NextLND)) ¬= CC) /* channel not found     */
     then do;
      call ErrorMsg(1,char(fixed(CC)),char(recnum),' ');
      if StartLND = 0                     /* initial case ANSI       */
       then
        return((1));                      /* default to first LND    */
       else
        return(LND_Nxt_spc(StartLND));    /* single space next line  */
     end;
   end/*while ¬ channel code*/;
   if Skip_Top                            /* a TOF was found         */
    then
     End_Page_Skip = true;                /* set global skip flag    */
   if (NextLND < FirstSkpLND)             /* went all the way around */
    then do;
     call Page_Environ(page_cnt,page_SKP); /* make a new page        */
     End_Page_Skip = false;               /* reset global skip flag  */
    end;
   return(NextLND);                       /* use this LND next       */
  end Find_Skip;
  Find_Spc: PROC(Spaces,StartLND) returns(fixed bin);
/*********************************************************************/
```

```
/* Finds LND to use for line spacing command.  If any LND in the     */
/* space chain from StartLND to the LND to be printed is flagged for */
/* spacing, the end-page spacing flag is set.                         */
/**********************************************************************/
  dcl
   Spaces         fixed bin(7),           /* channel code char value */
   Spc_Top        bit(1),                 /* save Spc form flag      */
   (StartLND, NextLND, i)
                  fixed bin(15);          /* LND array indexes       */
   if StartLND = 0                        /* initial case ANSI       */
    then
     NextLND = 1;                         /* start at first LND      */
    else
     NextLND = StartLND;                  /* start with current LND  */
   call Set_EndPage_Flags(NextLND);       /* reset starting flags    */
   Spc_Top = false;                       /* start already acted on  */
   do i = 1 to spaces;                    /* find space LND to use   */
    NextLND = LND_nxt_spc(NextLND);       /* follow the space chain  */
    call Set_EndPage_Flags(NextLND);      /* was this a TOF?         */
    if End_Page_Spc                       /* save page eject flag    */
     then
      Spc_Top = true;                     /* must space to next page */
   end/*i=1 to spaces*/;
   if Spc_Top                             /* a TOF was found         */
   /*******************************************************************/
   /* PSF's incorrect handling of spacing after page overflow is      */
   /* emulated by forcing print to LND 1 after an end of page while   */
   /* spacing.                                                        */
   /*******************************************************************/
    then
     if StartLND > NextLND                /* we've gone to a new page*/
      then do;
       call Page_Environ(page_cnt,page_SKP); /* make a new page      */
       NextLND = 1;                       /* start at top of page    */
       End_Page_Spc = false;              /* reset global SPC flag   */
      end;
      else                                /* we will go to a new page*/
       End_Page_Spc = true;               /* after this line prints  */
   return(NextLND);                       /* use this LND next       */
  end Find_Spc;
  Set_EndPage_Flags: Proc(LNDnum);
/**********************************************************************/
/* set global flags for end page skip/space from given LND            */
/**********************************************************************/
  dcl
   LNDnum              fixed bin(15);     /* lnd to set flags from   */
   End_Page_Skip = ((unspec(LND_flags(LNDnum)) & LNDFLAG_endskip) =
                    LNDFLAG_endskip);      /* initial page skip flag */
   End_Page_Spc = ((unspec(LND_flags(LNDnum)) & LNDFLAG_endspc) =
                   LNDFLAG_endspc);     ; /* initial page space flag */
  end Set_EndPage_Flags;
  Bracket: Proc(sfld,seq_num);
  /*******************************************************************/
  /* Write structured field begin/end pair type using seq_num        */
  /* for the sequence number and token value.                        */
  /*******************************************************************/
/*==> Replace with equivalents, if any, to emit non-AFP datastream.  */
  /*******************************************************************/
   dcl sfld            bit(24),           /* structured field value  */
   seq_num        fixed dec(07);          /* page num for seq. field */
    AFPout.len = AFPHdrLen + 8;           /* cc+afp header+id        */
    AFPout.count = AFPout.len - 1;        /* exclude cc              */
    AFPout.cc = ctl_5a;                   /* AFP carriage control    */
    AFPout.type = sfld;                   /* bracket field to write  */
    AFPout.flag = '0'b;                   /* clear flags             */
    AFPout.sequence = seq_num;            /* use page for seq. numb. */
```

```
   AFPout.data = substr(seq_num,3,8);    /* identify by page number */
   call Write_AFP(OutRec_ptr,0,(false)); /* pass to write routine   */
  end Bracket;
  Page_Environ: Proc(page_num,page_type);
/*********************************************************************/
/* Constructs AFP data for new page using data from the current data */
/* map.  An Active Environment Group is constructed from the page    */
/* map pointed to by the PFMTPTR. Page_num is the page number as     */
/* well as the flag for start and end document.  Global page number  */
/* is incremented for each new page.                                 */
/* If IDM structured fields and form skip carriage controls are      */
/* intermixed, adjacent controls only create new pages using the IDM.*/
/* if IMMs are found, the current page is ended, the input IMM is    */
/* issued, then a new page started.                                  */
/*********************************************************************/
/*==> Replace with code to start a new page, frame, screen, or       */
/* similar construct. Note previous page is ended, then a new one    */
/* started.  The proper PAGEDEF has already been selected.           */
/*********************************************************************/

  dcl page_num      fixed dec(07),        /*  page num/action flag   */
     page_type      bit(8) aligned,       /*  SKP = 1b (skip control)*/
                                          /* IDM = 10b               */
                                          /* IMM = 11b               */
   lptr           ptr,                   /* points to pagedef recs  */
   i              fixed bin(15),         /* loop counter            */
   seq_num        fixed dec(07),         /* page num for seq. field */
   (dt,tm,time_stamp) char(30) var static;/* identifying strings     */
  Comment: Proc(Text);
  /*******************************************************************/
  /* Write structured field NOP using page number for the sequence   */
  /* and Text for the data.                                          */
  /*******************************************************************/
   dcl Text            char(*);          /* text for NOP data       */
    AFPout.len = AFPHdrLen + length(Text); /*      nop length       */
    AFPout.count = AFPout.len - 1;        /* exclude cc              */
    AFPout.cc = ctl_5a;                   /* AFP carriage control    */
    AFPout.type = SF_NOP;                 /* begin document          */
    AFPout.flag = '0'b;                   /* clear flags             */
    AFPout.sequence = seq_num;            /* doc start sequence zero */
    AFPout.data = Text;                   /* NOP string              */
    call Write_AFP(OutRec_ptr,0,(false)); /* pass to write routine   */
  end Comment;
   /*****************************************************************/
   /* Determine if a new page environment should be written.  If the */
   /* record is an IDM, it is always written.  If it is a form skip, */
   /* it is written if it is not adjacent to an IDM.  Thus, only 1   */
   /* new page is created when IDMs and form skips are intermixed.   */
   /*****************************************************************/
   if page_type = page_text              /* force a new page        */
    then do;
     Pg_Pend = false;                    /* reset any pending page  */
    end;
    else do;                             /* a skip form request     */
     Pg_Pend = true;                     /* wait for IDM or text    */
     return;                             /* don't write yet         */
    end/*if not page pending*/;
   /* first, end the old page and start the next page.              */
   if page_num > 0                       /* not first page, so end  */
    then do;                             /* previous page.          */
     seq_num = page_num;                 /* use page number         */
     if BPT_Open                         /* close off present text  */
      then
       call Bracket(SF_EPT,seq_num);     /* end presentation text   */
     BPT_Open = false;                   /* text block closed       */
     call Bracket(SF_EPG,seq_num);       /* end page                */
```

```
  if held_IMM                          /* pass the IMM through    */
   then do;
    call Write_AFP(addr(Last_IMM),0,(false)); /* last seen IMM rec*/
    held_IMM = false;                  /* disposed of saved IMM   */
   end;
 end;
 else
  if page_num = 0
   then do;                            /* first page setup        */
    seq_num = 0;                       /* document starts at zero */
    call Bracket(SF_BDT,seq_num);      /* begin document          */
    dt = date;                         /* get date, time only once*/
    tm = time;                         /* to minimize system calls*/
    time_stamp = pgmname || substr(dt,1,2) || '//' || /*id string  */
                 substr(dt,3,2) || '/' || substr(dt,5,2) ||
                 ' ' || substr(tm,1,2) || ':' || substr(tm,3,2) ||
                 ':' || substr(tm,5,2) || ':' || substr(tm,7,3);
    call Comment(time_stamp);          /* write a NOP identifier  */
    if held_IMM                        /* pass the IMM through    */
     then do;
      call Write_AFP(addr(Last_IMM),0,(false)); /* initial IMM    */
      held_IMM = false;                /* disposed of saved IMM   */
    end;
   end/*start of document: page_num = 0*/;
   else do;                            /* end of document bracket */
    seq_num  = page_cnt;               /* final page w/last pg num*/
    if BPT_Open                        /* close off present text  */
     then do;
      call Bracket(SF_EPT,seq_num);    /* end presentation text   */
      BPT_Open = false;                /* text block closed       */
    end;
    call Bracket(SF_EPG,seq_num);      /* end page                */
    seq_num  = 0;                      /* reset so brackets match */
    AFPout.len = AFPHdrLen + length(time_stamp); /* nop length    */
    AFPout.count = AFPout.len - 1;     /* exclude cc              */
    AFPout.cc = ctl_5a;                /* AFP carriage control    */
    AFPout.type = SF_NOP;              /* begin document          */
    AFPout.flag = '0'b;                /* clear flags             */
    AFPout.sequence = 0;               /* doc start sequence zero */
    AFPout.data = time_stamp;          /* identify document       */
    call Write_AFP(OutRec_ptr,0,(false));  /*pass to write routine*/
    call Bracket(SF_EDT,seq_num);      /* end   document          */
    return;                            /* don't start another doc */
   end/*end of document: page_num < 0*/;
page_cnt = page_cnt + 1;               /* global page counter     */
seq_num = page_cnt;                    /* current page for bracket*/
/* mark the start of the new page.                               */
call Bracket(SF_BPG,seq_num);          /* begin page              */
call Comment(PFMT_Name);               /* identify map name       */
/* then,  write the BAG for the new environment.                 */
call Bracket(SF_BAG,seq_num);          /* begin environment       */
/* next, write any font or segment identifier records            */
if CHARS & (PFMT_MCF_cnt = 0)          /* use compatibility TRC?  */
 then do;
  MCF_record.sequence = seq_num;       /* identify with page numb */
  call Write_AFP(MCF_Rec_ptr,0,(false)); /* use length as string  */
 end/*if CHARS*/;
 else do;                              /* use MCF from PAGEDEF    */
  PDEFPTR = PFMT_1st_MCF;              /* initial MCF, if any     */
  do i = 1 to PFMT_MCF_cnt;            /* write each MCF          */
   lptr = addr(PDEF_Length);           /* start of record in list */
   lptr->AFPout.sequence = seq_num;    /* identify with page numb */
   call Write_AFP(lptr,0,(false));     /*  use length as string   */
   PDEFPTR = PDEF_next;                /* next PDEF rec   in list */
  end;
 end/*not CHARS*/;
```

```
  PDEFPTR = PFMT_1st_MPS;               /* initial MPS, if any     */
  do i = 1 to PFMT_MPS_cnt;             /* write each MPS          */
   lptr = addr(PDEF_Length);            /* start of record in list */
   lptr->AFPout.sequence = seq_num;     /* identify with page numb */
   call Write_AFP(lptr,0,(false));      /*  use length as string   */
   PDEFPTR = PDEF_next;                 /* next PDEF rec   in list */
  end;
  /* now search for the Page Descriptor                           */
  PDEFPTR = PFMT_Start;                 /* 1st rec of this datamap */
  InRec_ptr = addr(PDEF_data);          /* data portion of list    */
  do until (AFPDSREC.Type = SF_PGD);    /* find required PGD       */
   PDEFPTR = PDEF_next;                 /* next in list            */
   InRec_ptr = addr(PDEF_data);         /* data portion of list    */
  end;
  lptr = addr(PDEF_Length);             /* start of record in list */
  lptr->AFPout.sequence = seq_num;      /* identify with page numb */
  call Write_AFP(lptr,0,(false));       /* make fake string        */
  /* now search for Composed Text Control for compatibility only   */
  PDEFPTR = PFMT_Start;                 /* 1st rec of this datamap */
  InRec_ptr = addr(PDEF_data);          /* data portion of list    */
  do until (AFPDSREC.Type = SF_CTC);    /* find required CTC       */
   PDEFPTR = PDEF_next;                 /* next in list            */
   InRec_ptr = addr(PDEF_data);         /* data portion of list    */
  end;
  lptr = addr(PDEF_Length);             /* start of record in list */
  lptr->AFPout.sequence = seq_num;      /* identify with page numb */
  call Write_AFP(lptr,0,(false));       /* make fake string        */
  /* now search for the Presentation Text Descriptor               */
  PDEFPTR = PFMT_Start;                 /* 1st rec of this datamap */
  InRec_ptr = addr(PDEF_data);          /* data portion of list    */
  do until (AFPDSREC.Type = SF_PTD);    /* find required PTD       */
   PDEFPTR = PDEF_next;                 /* next in list            */
   InRec_ptr = addr(PDEF_data);         /* data portion of list    */
  end;
  lptr = addr(PDEF_Length);             /* start of record in list */
  lptr->AFPout.sequence = seq_num;      /* identify with page numb */
  call Write_AFP(lptr,0,(false));       /* make fake string        */
  /* finally, write the enclosing EAG bracket.                    */
  call Bracket(SF_EAG,seq_num);         /* end   environment       */
 end Page_Environ;
 Present_Text: Proc(LNDstart,TextData,NoPrint);
/*********************************************************************/
/* Constructs AFP record for presentation text from the current data */
/* map LND and the input text line.  LNDstart is the 1st LND to use  */
/* (reuse chains may use subsequent LNDs). TextData is the input     */
/* text string pointer.                                              */
/*********************************************************************/
/*==> The following procedures generate the text using data in the   */
/*==> LND array to control placement.  Strings other than PTXs could */
/*==> be constructed.  Be careful of text length and any limits of   */
/*==> the target device. This version does not optimize text.        */
/*********************************************************************/
 dcl
  (LNDstart, LND)     fixed bin(15),    /* lnd subscripts          */
  TextData            char(*),          /* input text to print     */
  NoPrint             bit(1),           /* text not to be printed  */
  Last_Orient         char(4),          /* orientation change flag */
  Last_FontID         char(1),          /* font change flag        */
  Last_Color          char(2),          /* color changed flag      */
  endstr              char(2),          /* text string terminator  */
  (PTXlen, LStart)    fixed bin(15);    /* text length, start pos  */
  PTX_Hdr: Proc;
 /*********************************************************************/
 /* Create the header for the PTX record. Length is set up after the */
 /*  entire record is generated. The initial escape is placed in the */
 /*  data output string and initial length set.                       */
```

```
/*********************************************************************/
 dcl
  WorkStr             char(2);           /* work string storage     */
   if ¬BPT_Open                          /* start new text object?  */
    then
     call Bracket(SF_BPT,Page_Cnt);      /* create begin text       */
   BPT_Open = true;                      /* tell all in text block  */
   AFPout.cc = ctl_5a;                   /* AFP control character   */
   AFPout.type = unspec(SF_PTX);         /* presentation text rcd.  */
   AFPout.flag = '0'b;                   /* flag byte cleared       */
   AFPout.sequence = Page_cnt;           /* use page for seq. numb. */
   unspec(workstr) = TC_ESC;
   substr(AFPout.data,1,2) = workstr;    /*initial ESC sequence     */
   PTXlen = 2;                           /* ESC length              */
  end PTX_Hdr;
  Concat_Txt: Proc(LND);
/*********************************************************************/
/* Create the text data stream and its associated text controls from*/
/*  the LND. Each call begins at the end of the current data output */
/*  string. Both input text and fixed text are created.  The global */
/*  string length, PTXlen, reflects the new length.                 */
/*********************************************************************/
  dcl
   LND           fixed bin(15),          /* LND for text controls   */
   len_byte      char(1),                /* 1-byte control length   */
   supp_id       char(1),                /* text suppress identifier*/
   Font_val      char(1),                /* hold area for font id   */
   WorkStr       char(256),              /* work string storage     */
   (curlen,                              /* amt. to print this TRN  */
    start,                               /* initial string position */
    i,                                   /* loop counter            */
    remain)      fixed bin(15),          /* left to print this line */
  maxtxt         fixed bin(15) init(255);/* maximum TRN text length */
 TRN_len: Proc(TRNlen);
 dcl TRNlen      fixed bin(15);          /* data length for TRN     */
 /*********************************************************************/
 /* Puts length TRNlen and TRN control into AFPout.Data at the     */
 /* current PTXlen position and increments PTXlen to the next slot.*/
 /*********************************************************************/
  unspec(len_byte) = substr(bit((TRNlen+2)),9,8);
               /* convert to length byte: length + control length */
  unspec(workstr) =                      /* TRN length, control code*/
         unspec((TC_TRN | TC_CCTL));
  substr(workstr,1,1) = len_byte;        /* stuff in length byte    */
  substr(AFPout.Data,PTXlen+1) = workstr;   /* concat to output str*/
  PTXlen = PTXlen + 2;                   /* bump by len of control  */
 end TRN_len;
 Font_ID: Proc returns(char);            /* function to get font id */
 /*********************************************************************/
 /* Determine the local font identifier to use.                     */
 /* if the LND font id is indicated in the LND, that font is used, */
 /* otherwise if TRC is specified and a  CHARS= parameter was given*/
 /* the TRC font is selected from the CHARS font list (or defaulted*/
 /* to the first if the TRC value is greater than the CHARS list). */
 /* If no CHARS= was specified, the corresponding value from the   */
 /* PAGEDEF MCF is used (or defaulted).  If there is no MCF, then  */
 /* the hardware default font is indicated.                         */
 /*********************************************************************/
 dcl
   FontID      char(1);                  /* hold temp font ident    */
 if (unspec(LND_Flags(LND)) &            /* use font from PAGEDEF?  */
     LNDFLAG_FontChg)
  then do;
   FontID = LND_FontID(LND);             /* use font from LND       */
  end;
  else do;                               /* TRC or defaults         */
```

```
 if TRC
  then do;
   FontID = Lineptr->LineRec.TRC_byte; /*     font index value   */
   FontID = translate(FontID,font_ids,
                      '0123456789'); /* map any compat TRCs     */
                    /* non-compatibility TRCs remain unchanged. */
   unspec(FontID) = fixed(unspec(FontID) + 1,8);
                                       /* fontids begin at 1   */
  if CHARS                           /* JCL CHARS= statement    */
   then
    if unspec(FontID) > substr(unspec(TRC_Char_Font_cnt),9,8)
                                      /* TRC ¬ in CHARS        */
     then do;
      unspec(FontID) = '00000001'b;   /* default to first font   */
      call ErrorMsg(2,' ',' ',' ');   /* inform user & continue  */
     end;
     else;                           /*use font in CHARS and TRC*/
   else                              /* no CHARS                */
    if PFMT_MCF_cnt > 0              /* MCF in PAGEDEF?         */
     then
      if unspec(FontID) > substr(unspec(TRC_Font_cnt),9,8)
       then do;
        FontID = low(1);             /* assume 1st fontid is 1  */
        call ErrorMsg(2,' ',' ',' '); /* inform user & continue  */
       end;
      else;                          /* use TRC in MCF          */
     else                            /* no MCF either           */
      FontID = high(1);              /* use hardware default    */
  end/*if TRC*/;
  else                               /* no TRC                  */
   if PFMT_MCF_cnt > 0 | CHARS       /* MCF in PAGEDEF or from  */
    then                             /* CHARS= (compatibility)  */
     unspec(FontID) = '00000001'b;   /* use 1st font in MCF     */
    else                             /* no MCF in PAGEDEF       */
     FontID = high(1);               /* use hardware default    */
 end;                                /* don't use PAGEDEF font  */
 return(FontID);                     /* send result back        */
end Font_ID;
 if (unspec(LND_Flags(LND)) &        /* text suppression?       */
     LNDFLAG_Suppress)
  then do;
   unspec(supp_id) = '00000001'b;    /* init to x'01' default   */
   if MSUPTR ¬= NULL                 /* was there an MSU record?*/
    then
     do i = 1 to MSU_Num_gps;        /* search for supp. token  */
      if MSU_Supp_Token(i) = LND_Supp_Token(LND)
       then do;                      /* token found--get id     */
        supp_id = MSU_Supp_ID(i);    /* save suppression id     */
        leave;                       /* and exit search         */
       end;
     end/*i=1 to num gps*/;
   unspec(workstr) =                 /* insert begin suppression*/
     unspec(TC_BSU | TC_CCTL) ||
     unspec(supp_id);                /* use id or 1 if no MSU   */
   substr(AFPout.Data,PTXlen+1) = workstr; /* concat to output str*/
   PTXlen = PTXlen + substr(TC_BSU,1,8); /*length of BSU sequence */
  end/*suppression flagged*/;
 if LND_Orient(LND) ¬= Last_Orient   /* generate only if changed*/
  then do;
   unspec(workstr) =                 /* start after last char   */
     unspec(TC_STO | TC_CCTL)  ||    /*text orient chained      */
     unspec(LND_Orient(LND));        /* value from LND          */
   substr(AFPout.Data,PTXlen+1) = workstr; /* concat to output str*/
   PTXlen = PTXlen + substr(TC_STO,1,8); /*length of STO sequence */
   Last_Orient = LND_Orient(LND);    /* remember last value     */
  end;
```

```
if (unspec(LND_Flags(LND)) &          /* set baseline indicated? */
    LNDFLAG_Baseline)
 then do;
  unspec(workstr) =                   /* start after last char   */
    (unspec((TC_AMB | TC_CCTL)) ||    /* abs move baseline   chnd*/
    unspec(LND_Baseln(LND)));         /* value from LND          */
  substr(AFPout.Data,PTXlen+1) = workstr; /* concat to output str*/
  PTXlen = PTXlen + substr(TC_AMB,1,8); /*length of AMB sequence */
 end;
if (unspec(LND_Flags(LND)) &          /* set inline   indicated? */
    LNDFLAG_InLine)
 then do;
  unspec(workstr) =                   /* start after last char   */
    (unspec((TC_AMI | TC_CCTL)) ||    /*abs move inline chned    */
    unspec(LND_InLine(LND)));         /* value from LND          */
  substr(AFPout.Data,PTXlen+1) = workstr; /* concat to output str*/
  PTXlen = PTXlen + substr(TC_AMI,1,8); /*length of AMI sequence */
 end;
if (unspec(LND_Flags(LND)) &          /* color indicated?        */
    LNDFLAG_Color)
 then
  if LND_COLOR(LND) ¬= last_Color     /* only if color changed   */
   then do;
    unspec(workstr) =                 /* color has changed       */
      (unspec((TC_STC | TC_CCTL)) ||  /* color change command    */
      unspec(LND_COLOR(LND)) ||       /* value from LND for color*/
      '00000001'b);                   /* default if not supported*/
    substr(AFPout.Data,PTXlen+1) = workstr; /*concat to out str*/
    PTXlen = PTXlen + substr(TC_STC,1,8); /*length STC   sequence*/
    Last_COLOR = LND_COLOR(LND);      /* remember last value     */
   end;
Font_val = Font_ID;                   /* get local font id       */
if Font_val ¬= last_FONTID            /* change font indicated?  */
 then do;
  unspec(workstr) =                   /* start after last char   */
    (unspec((TC_SCFL | TC_CCTL)) ||       /*text FONTID chained */
    unspec(Font_val));                /* value from LND          */
  substr(AFPout.Data,PTXlen+1) = workstr; /* concat to output str*/
  PTXlen = PTXlen + substr(TC_SCFL,1,8); /* length SCFL sequence */
  Last_FontID = Font_val;             /* remember last value     */
 end;
 if NoPrint                           /* text position only      */
  then
   return;                            /* don't generate any text */
 start = LND_Data_Start(LND) + 1;     /* begin text + substr bias*/
 remain = LND_Data_Len(LND);          /* remaining text length   */
 if (unspec(LND_Flags(LND)) &         /* is text from input?     */
     LNDFLAG_FixData) = 0
  then do;
   if remain < 0                      /* -1 denotes all remaining*/
    then
     remain = line_len;               /* maximum possible text   */
   remain = min(remain,(line_len-(start-1+Lstart)));
   /*for input text, the amount to print is the lesser of the    */
   /*LND text length, the input line length from the current     */
   /*starting point, or the greater of the input length - start  */
   /*point  if maximum length is specified in the LND (denoted   */
   /*by lnd data length of -1).                                  */
  end;
 do while(remain > 0);                /* place all text          */
  if (unspec(LND_Flags(LND)) &        /* determine text source   */
      LNDFLAG_FixData)
   then do;                           /* text from fixed data    */
    curlen = min(remain,maxtxt);      /* most that will fit TRN  */
    call TRN_len(curlen);             /* insert length, control  */
    substr(AFPout.Data, PTXlen+1) =         /* TRN from FDX      */
```

```
            substr(PFMT_1st_FDX->PDEF_Data,start+AFPHdrLen,curlen);
        end;
       else do;                          /* text from input record  */
        curlen =  min(remain,(Line_len-(start-1+Lstart)),maxtxt);
                                         /* residual text length    */
        call TRN_len(curlen);            /* insert length, control  */
        substr(AFPout.Data, PTXlen+1) =  /*      TRN from input text*/
            substr(LineData,start+Lstart,curlen);
        end;
      PTXlen = PTXlen + curlen;          /* running record length   */
      start = curlen + start;            /* begin in source string  */
      remain = remain - curlen;          /* residual text length    */
     end/*while remain > 0)*/;
     if (unspec(LND_Flags(LND)) &        /* text suppression?       */
         LNDFLAG_Suppress)
     then do;
      unspec(workstr) =                  /* insert end   suppression*/
        unspec(TC_ESU | TC_CCTL) ||
        unspec(supp_id);                 /* use id or 1 if no MSU   */
      substr(AFPout.Data,PTXlen+1) = workstr; /* concat to output str*/
      PTXlen = PTXlen + substr(TC_ESU,1,8); /*length of ESU sequence */
     end/*suppression flagged*/;
   end Concat_Txt;
  /*********************************************************************/
  /* Presentation text main routine.  Determines starting point for  */
  /* input text and initial font id (from TRC or LND), and calls the */
  /* routines to set up the text record and fill in as much text as  */
  /* is required.                                                    */
  /*********************************************************************/
  LND = LNDStart;                        /* initial LND for text    */
  if NOCC                                /* adjust for carr. ctl.   */
   then
    LStart = 0;                          /* 1st byte is printable   */
   else
    if TRC                               /* skip over TRC byte?     */
     then do;
      LStart = 2;                        /* skip TRC and carr. ctl. */
     end;
     else do;
      LStart = 1;                        /* carriage control only   */
     end;
   call PTX_Hdr;                         /* start AFP text record   */
   Last_Orient = '';                     /* no previous orientation */
   Last_FontID = '';                     /* no previous font        */
   if (unspec(LND_Flags(LND)) &          /* reusing input record?   */
       LNDFLAG_Reuse)
    then
     do until(LND = 0);                  /*follow reuse chain to end*/
      call Concat_Txt(LND);              /* get text for this LND   */
      LND = LND_Nxt_Reuse(LND);          /* next in reuse chain     */
     end;
    else
     call Concat_Txt(LND);               /* place text in output    */
   unspec(endstr) = '00000010'b ||
                    substr(TC_NOP,9,8);/*NOP len, no chain */
   substr(AFPout.Data,PTXlen+1) = endstr; /* NOP is  last control    */
   PTXlen = PTXlen + 2;                  /* adjust for NOP          */
   AFPout.Len = PTXlen + AFPHdrLen;      /* actual record length    */
   AFPout.Count = AFPout.Len - 1;        /* strct. fld len - cc     */
  end Present_Text;
/*********************************************************************/
/* Read each line, find the LND, format the line using the LND       */
/* values into an AFP data stream written to the AFPDS output file.  */
/* Machine, ANSI, and no channel codes are processed.                */
/*                                                                   */
/*********************************************************************/
```

```
/* initialize the font lookup table, since PL/I 1.5 can't handle    */
/* specification of hex values.                                     */
do i = 1 to 10;   /* convert integer to 1-byte numeric value 0-9    */
 unspec(addr(font_ids)->font_ids_tbl(i)) = substr(bit((i-1)),9,8);
end;
OPEN FILE (LineIn);
ON ENDFILE (LineIn) LineIn_EOF = true;
OPEN FILE(AFPDS) OUTPUT ;
alloc AFPout;                           /* output AFP data buffer  */
PFMTPTR = PFMT_List_Anchor;             /* set initial data map    */
linedata_started = false;
if ANSI_cc                              /* initial conditions      */
 then do;
  currlnd = 0;                          /* ANSI starts before 1st  */
  End_Page_Spc = false;                 /* can't be page overflow  */
 end;
 else do;
  currlnd = 1;                          /* machine starts with 1st */
  call Set_EndPage_Flags(currlnd);      /* initial space/skip      */
 end;
page_cnt = 0;                           /* first page flag         */
lineptr = Get_Line;                     /* read priming record     */
BPT_Open = false;                       /* no text yet generated   */
if ((lineptr ¬= null) &                 /* no first structured fld?*/
    ((lineptr->LineRec.cc ¬= ctl_5a) | NOCC))
 then do;
  call Page_Environ(0,page_text);       /* initial page setup      */
   linedata_started = true;
   end;
do while (lineptr ¬= null);             /* until all records read  */
 chnl_code = lineptr->LineRec.cc;       /* dereference channel code*/
 select;                                /* by carriage ctl type    */
  when (chnl_code = ctl_5a & ¬NOCC) do;  /* structured field recs   */
  /*******************************************************************/
  /* Structured field records that are not IDM are passed to the    */
  /*output unaltered.  If carriage controls are not used, then      */
  /*'5A' records are treated as data and printed.                    */
  /*******************************************************************/
   select (lineptr->LineAFP.type);      /* type of structured field*/
    when (SF_IDM) do;                   /* invoke data map         */
     DataMap = substr(lineptr->LineAFP.rest,1,8); /* data map name  */
     if DataMap ¬= PFMT_Name            /* not current format?     */
      then do;
       PFMTPTR = PFMT_List_Anchor;      /* start from first map    */
       do while (¬(DataMap = PFMT_Name | /* matching name or        */
                 PFMT_Next = null));    /* list end-name not found */
        PFMTPTR = PFMT_Next;            /* next in map list        */
       end;                             /* while not end of chain  */
      end;                              /* datamap ¬= pfmt_name    */
     if DataMap = PFMT_Name             /* found the data map      */
      then do;
       currlnd = 1;                     /* start new page with     */
       call Page_Environ(page_cnt,page_IDM); /* first LND in map    */
   linedata_started = true;
      end;                              /* datamap found           */
      else do;                          /* map not found-error     */
       call Page_Environ(-1,page_text);  /* write ending doc. envir.*/
       /*terminate if unknown map requested in line input.          */
       call ErrorExit(9,DataMap,' ',' '); /* exit via    error rtn  */
      end;                              /* datamap not found-quit  */
    end;                                /* when IDM                */
    when (SF_IMM) do;                   /* invoke medium map       */
     /* IMM must end current page, insert IMM, then start new page. */
     Last_IMM = LineData;               /* save for deferred write */
     held_IMM = true;                   /* we're saving an IMM     */
     call Page_Environ(page_cnt,page_IMM); /* newpage does it all   */
```

```
  end;                                  /* when IMM               */
  otherwise do;                         /* any other SF record    */
  /* Assume structured field must not be in current presentation  */
  /* text environment, if one exists at this point.               */
   if BPT_Open
    then
     call Bracket(SF_EPT,page_cnt);     /* close current text     */
   BPT_Open = false;                    /* tell all block closed  */
   call Write_AFP(Lineptr,0,(false));   /* write 5a & reset envir. */
  end/*all others*/;
 end/*select AFP type*/;
end/*select structured field*/;
when (mach_cc) do;                      /* machine carriage control*/
/*********************************************************************/
/*Machine carriage controls write and then position (to the next  */
/*LND) for either skipping or spacing.  Control only codes create */
/*text positioning commands so that mixed 5A data can be          */
/*positioned correctly.                                            */
/*********************************************************************/
 if linedata_started = false then do;
    call Page_Environ(0,page_text);
    linedata_started = true;
    end;
 if (chnl_code & mach_immed) = 0        /* after write (not immed) */
  then
   call Write_AFP(OutRec_ptr,currlnd,(false)); /*write  data first*/
  else
   if ¬(End_Page_Skip | End_Page_Spc)   /* position within page    */
    then
     call Write_AFP(OutRec_ptr,currlnd,(true));  /*just position  */
 select;                                /* machine carriage control*/
  when (chnl_code & mach_skip) do;      /*machine skip channel     */
   if End_Page_Skip
    then
     call Page_Environ(page_cnt,page_SKP);   /* set next pg envir.*/
   ChNum = '0'b;                        /* init for substring      */
   substr(ChNum,5,4) = substr(chnl_code,2,4); /*extract chann. num*/
   currlnd = Find_Skip(ChNum,currlnd); /*LND for skip channel     */
  end/*machine skip to channel*/;
  when ((chnl_code & mach_skip) = 0) do;      /*machine space     */
   if End_Page_Spc
    then do;
     call Page_Environ(page_cnt,page_SKP);   /* set next pg envir.*/
    end;
   space_cnt = fixed((chnl_code / 8)); /* number of spaces        */
   currlnd = Find_Spc(space_cnt,currlnd);   /*LND for space/print */
  end/*machine space*/;
  otherwise do;                         /* unrecognized control    */
   call ErrorMsg(1,char(chnl_code),char(recnum),' ');
  end/*unrecognized control*/;
 end/*select skip or space*/;
end/*select machine control*/;
when (ansi_cc) do;                      /* ANSI carriage    control*/
/*********************************************************************/
/* ANSI carriage controls are processed in the opposite order of  */
/*machine controls. Thus, the data is written with the current LND*/
/*and then the next LND to use is determined.  Page skip/space    */
/*breaks are set up for the next data line to print.               */
/*********************************************************************/
 if linedata_started = false then do;
    call Page_Environ(0,page_text);
    linedata_started = true;
    end;
 select (chnl_code);                    /* ANSI carriage    control*/
  when (ANSI_skc1, ANSI_skc2, ANSI_skc3,      /*ANSI skip channel */
        ANSI_skc4, ANSI_skc5, ANSI_skc6,
```

```
          ANSI_skc7, ANSI_skc8, ANSI_skc9,
          ANSI_skcA, ANSI_skcB, ANSI_skcC) do;
      if End_Page_Skip
       then
        call Page_Environ(page_cnt,page_SKP); /*set next pg envir.*/
      unspec(ChCode) = chnl_code;         /*convert to char for index*/
      unspec(ChNum) = fixed(index('123456789ABC',
                      ChCode),8);                  /*find chanl number*/
      currlnd = Find_Skip(ChNum,currlnd); /*LND for skip channel     */
     end/*ANSI skip to channel*/;
     when (ANSI_spc0, ANSI_spc1,                        /*ANSI space */
          ANSI_spc2, ANSI_spc3) do;
      if End_Page_Spc
       then do;
        call Page_Environ(page_cnt,page_SKP);   /* set next pg envir.*/
       end;
      unspec(ChCode) = chnl_code;         /*convert to char for index*/
      space_cnt = index('+ 0-',ChCode) - 1; /* # of spaces,+=none    */
      currlnd = Find_Spc(space_cnt,currlnd); /*LND spacing          */
     end/*ANSI space*/;
     otherwise do;                        /* unrecognized control    */
      call ErrorMsg(1,char(chnl_code),char(recnum),' ');
     end/*unrecognized control*/;
    end/*select skip or space*/;
    call Write_AFP(OutRec_ptr,currlnd,(false));     /*wrt after skip */
   end/*select ANSI*/;
   when (NoCC) do;                      /* no carriage control     */
    call Write_AFP(OutRec_ptr,currlnd,(false)); /* write line first  */
    if End_Page_Spc
     then
      call Page_Environ(page_cnt,page_SKP); /* set up for new page   */
    currlnd = Find_Spc(1,currlnd);        /*LND single spacing       */
   end/*no carriage control*/;
  end/*select carriage control*/;
  Lineptr = Get_line;                   /* next record             */
 end/*while not EOF linedata*/;
 close FILE (LineIn);
/*********************************************************************/
/* write terminating document bracket before close.                  */
/*********************************************************************/
 call Page_Environ(-1,page_text);        /* write ending doc. envir.*/
 close FILE(AFPDS);
end Process_LineData;
 ErrorMsg:  PROCEDURE(MSGIDX,S1,S2,S3);
 /*************************************************************/
 /***                                                       ***/
 /***  ERRORMSG displays the appropriate error message and  ***/
 /*** returns control to the caller.                        ***/
 /***                                                       ***/
 /*** eventually will get message from auxillary list and    ***/
 /*** either terminate, return control, or return error to   ***/
 /*** external caller (if called from API).                  ***/
 /***                                                       ***/
 /*************************************************************/
 dcl MSGIDX      FIXED BIN(15),
     (s1,s2,s3)  char(80) var,           /* substituted strings     */
     ERR_MSG(10) CHAR(240);              /* ERROR MESSAGE ARRAY     */
                /***********************************/
                /*   Error Messages Initialized    */
                /***********************************/
 ERR_MSG(1) = 'APS346I DATA IN AN INPUT RECORD OR PAGEDEF RESOURCE' ||
              ' IS INVALID: A SKIP TO A NON-EXISTANT CHANNEL = '    ||
              S1 || ' ON RECORD ' || S2 || ' WAS DETECTED WITHIN '  ||
              'THE LND STRUCTURED FIELDS. OUTPUT WAS FORCED TO '    ||
              'SINGLE SPACING AND MAY CONTAIN BLANK PAGES.';
 ERR_MSG(2) = 'APS341I A FONT NAMED IN THE PAGEDEF RESOURCE OVERRI' ||
```

```
            'DES THE FONT SPECIFIED IN THE TABLE REFERENCE'       ||
            'CHARACTER ON ONE OR MORE RECORDS. PRINTED OUTPUT '   ||
            'MAY BE ACCEPTABLE';
 put file(SYSPRINT) skip list (ERR_MSG(MSGIDX));
END ErrorMsg;
ErrorExit: PROCEDURE(MSGIDX,S1,S2,S3);
/*************************************************************/
/***                                                       ***/
/***  ErrorExit closes all files and the appropriate error  ***/
/***message is displayed.                                   ***/
/***                                                       ***/
/*** eventually will get message from auxillary list and    ***/
/*** either terminate, return control, or return error to   ***/
/*** external caller (if called from API).                  ***/
/***                                                       ***/
/*************************************************************/
dcl MSGIDX      FIXED BIN(15),
    (s1,s2,s3)  char(*)  var,           /* substituted strings     */
    ERR_MSG(10) CHAR(240);              /* ERROR MESSAGE ARRAY     */
              /***********************************/
              /*   Error Messages Initialized    */
              /***********************************/
ERR_MSG(1) = 'Invalid PAGEDEF more than 1 BPM found';
ERR_MSG(2) = 'Invalid PAGEDEF no BPM found.';
ERR_MSG(3) = 'Invalid PAGEDEF BAG without EAG.';
ERR_MSG(4) = 'Invalid PAGEDEF EAG without BAG.';
ERR_MSG(5) = 'APS216I AN INPUT-DATA RECORD IS MISSING: ' || S1 ||
             ' STRUCTURED FIELD WAS RECEIVED, BUT NO '   || S2 ||
             ' STRUCTURED FIELD WAS SPECIFIED.';
ERR_MSG(7) = 'APS305I DATA IN A PAGEDEF RESOURCE IS INVALID: '   ||
             'STRUCTURED FIELD ' || S1 || ' WAS FOUND WHERE AN ' ||
             'EDX STRUCTURED FIELD WAS EXPECTED';
ERR_MSG(8) = 'APS300I DATA IN A PAGEDEF RESOURCE IS INVALID: ' ||
             'THE NEXT LINE DESCRIPTOR IF SKIPPING PARAMETER ' ||
             'VALUE IN LND STRUCTURED FIELD NUMBER ' || S1 || ' IS 0';
ERR_MSG(9) = 'APS162I MISMATCH BETWEEN PRINT DATA SET AND PAGEDEF' ||
             ' RESOURCE: DATA MAP ''' || S1 || ''' SPECIFIED IN'   ||
             ' IDM STRUCTURED FIELD WAS NOT FOUND IN PAGEDEF '''   ||
             S2 || '''';
ERR_MSG(10)= 'APSTTTI INVALID CHARS= PARAMETER SPECIFIED.  '       ||
             S1 || ' IS NOT A VALID VALUE';
 put file(SYSPRINT) skip list (ERR_MSG(MSGIDX));
 call pliretc(16);   /*set return code*/
 STOP;
END ErrorExit;
END LN2AFPN;
```

A.6.2 Included PL/I Definitions

```
*%GOTO SFIDPLI;       /*                                              00010034
       MACRO                                                         00020000
       SFIDEQU                                                       00030000
*--------------------------------------------------------------------* 00040000
* SYMBOLIC EQUATES FOR STRUCTURED FIELD IDENTIFIERS AND COMPOSED-TEXT | 00050000
* CONTROLS.  SEE PSF DATA STREAM REFERENCE, SH35-0073-03.            | 00060000
*                                                                    | 00070000
*  LAST UPDATE ON 10 JAN 1990 AT 15:22:50 BY VEND730 VERSION 02      | 00080037
*   ADD NEW STRUCTURED FIELD TYPES FOR GRAPHICS, BAR CODES.          | 00090037
*  LAST UPDATE ON 2 JAN 1990 AT 15:24:53 BY VEND730 VERSION 02       | 00100037
*   USE BIT STRING VALUES FOR PLI CONTSTANTS FOR PL/I 1.5.           | 00110037
*--------------------------------------------------------------------* 00120000
* SYMBOLS FOR STRUCTURED FIELD IDENTIFIERS                           00130000
SF$FNI   EQU   X'D38C89',3,C'X'    * FONT INDEX                     00140037
SF$CFI   EQU   X'D38C8A',3,C'X'    * CODED FONT INDEX               00150037
SF$MCC   EQU   X'D3A288',3,C'X'    * MEDIUM COPY COUNT              00160037
SF$FNM   EQU   X'D3A289',3,C'X'    * FONT PATTERNS MAP              00170037
```

```
SF$OBD   EQU   X'D3A66B',3,C'X'    * OBJECT AREA DESCRIPTOR              00180037
SF$IID   EQU   X'D3A67B',3,C'X'    * IMAGE INPUT DESCRIPTOR              00190037
SF$CPD   EQU   X'D3A687',3,C'X'    * CODE PAGE DESCRIPTOR                00200037
SF$MDD   EQU   X'D3A688',3,C'X'    * MEDIUM DESCRIPTOR                   00210037
SF$FND   EQU   X'D3A689',3,C'X'    * FONT DESCRIPTOR                     00220037
SF$CTD   EQU   X'D3A69B',3,C'X'    * COMPOSED-TEXT DESCRIPTOR            00230037
SF$PTD   EQU   X'D3A69B',3,C'X'    * PRESENTATION TEXT DESCRIPTOR        00240037
SF$PGD   EQU   X'D3A6AF',3,C'X'    * PAGE DESCRIPTOR                     00250037
SF$GDD   EQU   X'D3A6BB',3,C'X'    * GRAPHICS DATA DESCRIPTOR            00260037
SF$FGD   EQU   X'D3A6C5',3,C'X'    * FORM ENVIRONMENT GROUP DESCRIPTOR   00270037
SF$DXD   EQU   X'D3A6E3',3,C'X'    * DATA MAP TRANSMISSION SUBCASE DESC  00280037
SF$LND   EQU   X'D3A6E7',3,C'X'    * LINE DESCRIPTOR                     00290037
SF$BDD   EQU   X'D3A6EB',3,C'X'    * BAR CODE DATA DESCRIPTOR            00300037
SF$IDD   EQU   X'D3A6FB',3,C'X'    * IMAGE DATA DESCRIPTOR IO            00310037
SF$IOC   EQU   X'D3A77B',3,C'X'    * IMAGE OUTPUT CONTROL                00320037
SF$CPC   EQU   X'D3A787',3,C'X'    * CODE PAGE CONTROL                   00330037
SF$MMC   EQU   X'D3A788',3,C'X'    * MEDIUM MODIFICATION CONTROL         00340037
SF$FNC   EQU   X'D3A789',3,C'X'    * FONT CONTROL                        00350037
SF$CFC   EQU   X'D3A78A',3,C'X'    * CODED FONT CONTROL                  00360037
SF$CTC   EQU   X'D3A79B',3,C'X'    * COMPOSED-TEXT CONTROL               00370037
SF$CCP   EQU   X'D3A7CA',3,C'X'    * CONDITIONAL PROCESSING CONTROL      00380037
SF$BPS   EQU   X'D3A85F',3,C'X'    * BEGIN PAGE SEGMENT                  00390037
SF$BIM   EQU   X'D3A87B',3,C'X'    * BEGIN IMAGE OBJECT IM               00400037
SF$BCP   EQU   X'D3A887',3,C'X'    * BEGIN CODE PAGE                     00410037
SF$BFN   EQU   X'D3A889',3,C'X'    * BEGIN FONT                          00420037
SF$BCF   EQU   X'D3A88A',3,C'X'    * BEGIN CODED FONT                    00430037
SF$BGR   EQU   X'D3A88B',3,C'X'    * BEGIN GRAPHICS OBJECT               00440037
SF$BCT   EQU   X'D3A89B',3,C'X'    * BEGIN COMPOSED-TEXT BLOCK           00450037
SF$BPT   EQU   X'D3A89B',3,C'X'    * BEGIN PRESENTATION TEXT             00460037
SF$BDT   EQU   X'D3A8A8',3,C'X'    * BEGIN DOCUMENT                      00470037
SF$BPG   EQU   X'D3A8AF',3,C'X'    * BEGIN PAGE                          00480037
SF$BDG   EQU   X'D3A8C4',3,C'X'    * BEGIN DOCUMENT ENVIRONMENT GROUP    00490037
SF$BFG   EQU   X'D3A8C5',3,C'X'    * BEGIN FORM ENVIRONMENT GROUP        00500037
SF$BRG   EQU   X'D3A8C6',3,C'X'    * BEGIN RESOURCE GROUP                00510037
SF$BOG   EQU   X'D3A8C7',3,C'X'    * BEGIN OBJECT ENVIRONMENT GROUP      00520037
SF$BAG   EQU   X'D3A8C9',3,C'X'    * BEGIN ACTIVE ENVIRONMENT GROUP      00530037
SF$BDM   EQU   X'D3A8CA',3,C'X'    * BEGIN DATA MAP                      00540037
SF$BPM   EQU   X'D3A8CB',3,C'X'    * BEGIN PAGE MAP                      00550037
SF$BMM   EQU   X'D3A8CC',3,C'X'    * BEGIN MEDIUM MAP                    00560037
SF$BFM   EQU   X'D3A8CD',3,C'X'    * BEGIN FORM MAP                      00570037
SF$BR    EQU   X'D3A8CE',3,C'X'    * BEGIN RESOURCE                      00580037
SF$BMO   EQU   X'D3A8DF',3,C'X'    * BEGIN MEDIUM OVERLAY                00590037
SF$BBC   EQU   X'D3A8EB',3,C'X'    * BEGIN BAR CODE OBJECT               00600037
SF$BDX   EQU   X'D3A8E3',3,C'X'    *BEGIN DATA MAP TRANSMISSION SUBCASE  00610037
SF$BIMO  EQU   X'D3A8FB',3,C'X'    * BEGIN IMAGE OBJECT IO               00620037
SF$EPS   EQU   X'D3A95F',3,C'X'    * END PAGE SEGMENT                    00630037
SF$EIM   EQU   X'D3A97B',3,C'X'    * END IMAGE BLOCK                     00640037
SF$ECP   EQU   X'D3A987',3,C'X'    * END CODE PAGE                       00650037
SF$EFN   EQU   X'D3A989',3,C'X'    * END FONT                            00660037
SF$ECF   EQU   X'D3A98A',3,C'X'    * END CODED FONT                      00670037
SF$ECT   EQU   X'D3A99B',3,C'X'    * END COMPOSED-TEXT BLOCK             00680037
SF$EPT   EQU   X'D3A99B',3,C'X'    * END PRESENTATION TEXT BLOCK         00690037
SF$EPG   EQU   X'D3A9AF',3,C'X'    * END PAGE                            00700037
SF$EDT   EQU   X'D3A9A8',3,C'X'    * END DOCUMENT                        00710037
SF$EGO   EQU   X'D3A9BB',3,C'X'    * END END GRAPHICS OBJECT             00720037
SF$EDG   EQU   X'D3A9C4',3,C'X'    * END DOCUMENT ENVIRONMENT GROUP      00730037
SF$EFG   EQU   X'D3A9C5',3,C'X'    * END FORM ENVIRONMENT GROUP          00740037
SF$ERG   EQU   X'D3A9C6',3,C'X'    * END RESOURCE GROUP                  00750037
SF$EAG   EQU   X'D3A9C9',3,C'X'    * END ACTIVE ENVIRONMENT GROUP        00760037
SF$EDM   EQU   X'D3A9CA',3,C'X'    * END DATA MAP                        00770037
SF$EPM   EQU   X'D3A9CB',3,C'X'    * END PAGE MAP                        00780037
SF$EMM   EQU   X'D3A9CC',3,C'X'    * END MEDIUM MAP                      00790037
SF$EFM   EQU   X'D3A9CD',3,C'X'    * END FORM MAP                        00800037
SF$ER    EQU   X'D3A9CE',3,C'X'    * END RESOURCE                        00810037
SF$EMO   EQU   X'D3A9DF',3,C'X'    * END MEDIUM OVERLAY                  00820037
SF$EDX   EQU   X'D3A9E3',3,C'X'    * END DATA MAP TRANSMISSION SUBCASE   00830037
```

```
SF$LNC   EQU   X'D3AAE7',3,C'X'    * LINE DESCRIPTOR COUNT              00840037
SF$EBC   EQU   X'D3A9EB',3,C'X'    * END BAR CODE OBJECT                00850037
SF$FDS   EQU   X'D3AAEC',3,C'X'    * FIXED DATA SIZE                    00860037
SF$MCF2  EQU   X'D3AB8A',3,C'X'    * MAP CODED FONT (FORMAT 2)          00870037
SF$MGO   EQU   X'D3ABBB',3,C'X'    * MAP GRAPHIC OBJECT                 00880037
SF$IDM   EQU   X'D3ABCA',3,C'X'    * INVOKE DATA MAP                    00890037
SF$IMM   EQU   X'D3ABCC',3,C'X'    * INVOKE MEDIUM MAP                  00900037
SF$MPO   EQU   X'D3ABD8',3,C'X'    * MAP PAGE OVERLAY                   00910037
SF$MSU   EQU   X'D3ABEA',3,C'X'    * MAP SUPPRESSION                    00920037
SF$MBC   EQU   X'D3ABEB',3,C'X'    * MAP BAR CODE                       00930037
SF$MIO   EQU   X'D3ABFB',3,C'X'    * MAP IO IMAGE OBJECT                00940037
SF$OBP   EQU   X'D3AC6B',3,C'X'    * OBJECT AREA POSITION               00950037
SF$ICP   EQU   X'D3AC7B',3,C'X'    * IMAGE CELL POSITION                00960037
SF$CPI   EQU   X'D3AC87',3,C'X'    * CODE PAGE INDEX                    00970037
SF$FNP   EQU   X'D3AC89',3,C'X'    * FONT POSITION                      00980037
SF$PGP   EQU   X'D3ACAF',3,C'X'    * PAGE POSITION                      00990037
SF$FNO   EQU   X'D3AE89',3,C'X'    * FONT ORIENTATION                   01000037
SF$IPS   EQU   X'D3AF5F',3,C'X'    * INCLUDE PAGE SEGMENT               01010037
SF$IPO   EQU   X'D3AFD8',3,C'X'    * INCLUDE PAGE OVERLAY               01020037
SF$MPS   EQU   X'D3B15F',3,C'X'    * MAP PAGE SEGMENT                   01030037
SF$MCF   EQU   X'D3B18A',3,C'X'    * MAP CODED FONT (FORMAT 1)          01040037
SF$MMO   EQU   X'D3B1DF',3,C'X'    * MAP MEDIUM OVERLAY                 01050037
SF$IRD   EQU   X'D3EE7B',3,C'X'    * IMAGE RASTER DATA                  01060037
SF$FNG   EQU   X'D3EE89',3,C'X'    * FONT PATTERNS                      01070037
SF$CTX   EQU   X'D3EE9B',3,C'X'    * COMPOSED-TEXT DATA                 01080037
SF$PTX   EQU   X'D3EE9B',3,C'X'    * PRESENTATION TEXT                  01090037
SF$GAD   EQU   X'D3EEBB',3,C'X'    * GRAPHICS DATA                      01100037
SF$BDA   EQU   X'D3EEEB',3,C'X'    * BAR CODE DATA                      01110037
SF$FDX   EQU   X'D3EEEC',3,C'X'    * FIXED DATA TEXT                    01120037
SF$NOP   EQU   X'D3EEEE',3,C'X'    * NO OPERATION                       01130000
SF$IPD   EQU   X'D3EEFB',3,C'X'    * IMAGE PICTURE DATA                 01140037
* COMPOSED TEXT CONTROL SEQUENCES.                                     01150000
*   TEXT CONTROLS ARE DESCRIBED AS FOLLOWS:                            01160000
*   BYTE 1: LENGTH. ZERO REPRESENTS A VARIABLE LENGTH CONTROL.         01170000
*   BYTE 2: TEXT-CONTROL CODE, UNCHAINED. (EVEN VALUES)                01180000
*           CHAINED CONTROL CODES ARE REPRESENTED BY THE LOW-ORDER      01190000
*           BIT TURNED ON (ODD VALUES).  THESE VALUES MAY BE CODED     01200000
*           SYMBOLICALLY BY: CHAINED_VALUE EQU "TC"+CCTL, WHERE        01210000
*           "TC" IS THE TEXT-CONTROL MNEMONIC.                         01220000
TC$AMB   EQU   X'04D2',2,C'X'      * ABSOLUTE MOVE BASELINE             01230000
TC$AMI   EQU   X'04C6',2,C'X'      * ABSOLUTE MOVE INLINE               01240000
TC$BLN   EQU   X'02D8',2,C'X'      * BEGIN LINE                         01250000
TC$BSU   EQU   X'03F2',2,C'X'      * BEGIN SUPPRESSION                  01260000
TC$CCTL  EQU   B'00000001'         * CHAINED CONTROL FLAG BIT           01270000
TC$DBR   EQU   X'07E6',2,C'X'      * DRAW BASELINE RUL                  01280000
TC$DIR   EQU   X'07E4',2,C'X'      * DRAW INLINE RUL                    01290000
TC$ESC   EQU   X'2BD3',2,C'X'      * ESCAPE SEQUENCE                    01300000
TC$ESU   EQU   X'03F4',2,C'X'      * END SUPPRESSION                    01310000
TC$NOP   EQU   X'00F8',2,C'X'      * NO OPERATION                       01320000
TC$RMB   EQU   X'04D4',2,C'X'      * RELATIVE MOVE BASELINE             01330000
TC$RMI   EQU   X'04C8',2,C'X'      * RELATIVE MOVE INLINE               01340000
TC$RPS   EQU   X'00EE',2,C'X'      * REPEAT STRING                      01350000
TC$SBI   EQU   X'04D0',2,C'X'      * SET BASELINE INCREMENT             01360000
TC$SCFL  EQU   X'03F0',2,C'X'      * SET CODED FONT LOCAL               01370026
TC$SII   EQU   X'04C2',2,C'X'      * SET INTERCHARACTER INCREMENT       01380000
TC$SIM   EQU   X'04C0',2,C'X'      * SET INLINE MARGIN                  01390000
TC$STC   EQU   X'0574',2,C'X'      * SET TEXT COLOR                     01400031
TC$STO   EQU   X'06F6',2,C'X'      * SET TEXT ORIENTATION               01410031
TC$SVI   EQU   X'04C4',2,C'X'      * SET VARIABLE SPACE CHAR INCREMENT  01420000
TC$TRN   EQU   X'00DA',2,C'X'      * TRANSPARENT DATA                   01430000
         MEND  ,                   */                                  01440000
 %SFIDPLI: ;                                                           01450000
  DECLARE                                                              01460000
 /* LAST UPDATE ON 18 AUG 1989 AT 10:30:14 BY VEND730 VERSION 01 */     01470000
 /*--------------------------------------------------------------*/     01480000
 /* SYMBOLIC EQUATES FOR STRUCTURED FIELD IDENTIFIERS AND        */     01490000
```

```
/* COMPOSED-TEXT CONTROLS. SEE PSF DATA STREAM REFERENCE,       */     01500000
/* SH35-0073-03.                                               */     01510000
/*-------------------------------------------------------------*/     01520000
/* SYMBOLS FOR STRUCTURED FIELD IDENTIFIERS:                   */     01530000
 SF_BAG  BIT(24) STATIC         /* BEGIN ACTIVE ENVIRON  D3A8C9 HEX */ 01540000
         INIT('110100111010100011001001'B),                          01550000
 SF_BBC  BIT(24) STATIC         /* BEGIN BAR CODE OBJECT D3A8EB HEX */ 01560037
         INIT('110100111010100011101011'B),                          01570037
 SF_BCF  BIT(24) STATIC         /* BEGIN CODED FONT      D3A88A HEX */ 01580000
         INIT('110100111010100010001010'B),                          01590000
 SF_BCP  BIT(24) STATIC         /* BEGIN CODE PAGE       D3A887 HEX */ 01600000
         INIT('110100111010100010000111'B),                          01610000
 SF_BCT  BIT(24) STATIC         /* BEGIN COMPOSED-TEXT   D3A89B HEX */ 01620037
         INIT('110100111010100010011011'B),                          01630037
 SF_BDA  BIT(24) STATIC         /* BAR CODE DATA         D3EEEB HEX */ 01640037
         INIT('110100111110111011101011'B),                          01650037
 SF_BDD  BIT(24) STATIC         /* BAR CODE DESCRIPTOR   D3A6EB HEX */ 01660037
         INIT('110100111010011011101011'B),                          01670037
 SF_BDG  BIT(24) STATIC         /* BEGIN DOC ENVIRON GRP D3A8C4 HEX */ 01680037
         INIT('110100111010100011000100'B),                          01690037
 SF_BDM  BIT(24) STATIC         /* BEGIN DATA MAP        D3A8CA HEX */ 01700000
         INIT('110100111010100011001010'B),                          01710000
 SF_BDT  BIT(24) STATIC         /* BEGIN DOCUMENT        D3A8A8 HEX */ 01720000
         INIT('110100111010100010101000'B),                          01730000
 SF_BDX  BIT(24) STATIC  /* BEGIN DATA MAP XMIT SUBCASE D3A8E3 HEX */  01740000
         INIT('110100111010100011100011'B),                          01750000
 SF_BFG  BIT(24) STATIC  /* BEGIN FORM ENVIRONMENT GRP  D3A8C5 HEX */  01760000
         INIT('110100111010100011000101'B),                          01770000
 SF_BFM  BIT(24) STATIC  /* BEGIN FORM MAP      D3A8CD HEX */          01780000
         INIT('110100111010100011001101'B),                          01790000
 SF_BFN  BIT(24) STATIC  /* BEGIN FONT          D3A889 HEX */          01800000
         INIT('110100111010100010001001'B),                          01810000
 SF_BGR  BIT(24) STATIC         /* BEGIN GRAPHICS OBJECT D3A88B HEX */ 01820037
         INIT('110100111010100010001011'B),                          01830037
 SF_BIM  BIT(24) STATIC         /* BEGIN IMAGE BLOCK     D3A87B HEX */ 01840037
         INIT('110100111010100001111011'B),                          01850037
 SF_BIMO BIT(24) STATIC         /* BEGIN IMAGE OBJECT IO D3A8FB HEX */ 01860037
         INIT('110100111010100011111011'B),                          01870037
 SF_BMM  BIT(24) STATIC         /* BEGIN MEDIUM MAP      D3A8CC HEX */ 01880000
         INIT('110100111010100011001100'B),                          01890000
 SF_BMO  BIT(24) STATIC         /* BEGIN MEDIUM OVERLAY  D3A8DF HEX */ 01900000
         INIT('110100111010100011011111'B),                          01910000
 SF_BOG  BIT(24) STATIC         /* BEGIN OBJ ENVIR GROUP D3A8C7 HEX */ 01920037
         INIT('110100111010100011000111'B),                          01930037
 SF_BPG  BIT(24) STATIC         /* BEGIN PAGE            D3A8AF HEX */ 01940000
         INIT('110100111010100010101111'B),                          01950000
 SF_BPM  BIT(24) STATIC         /* BEGIN PAGE MAP        D3A8CB HEX */ 01960000
         INIT('110100111010100011001011'B),                          01970000
 SF_BPS  BIT(24) STATIC         /* BEGIN PAGE SEGMENT    D3A85F HEX */ 01980000
         INIT('110100111010100001011111'B),                          01990000
 SF_BPT  BIT(24) STATIC     /* BEGIN PRESENTATION TEXT   D3A89B HEX */ 02000000
         INIT('110100111010100010011011'B),                          02010000
 SF_BR   BIT(24) STATIC         /* BEGIN RESOURCE        D3A8CE HEX */ 02020000
         INIT('110100111010100011001110'B),                          02030000
 SF_BRG  BIT(24) STATIC         /* BEGIN RESOURCE GROUP  D3A8C6 HEX */ 02040000
         INIT('110100111010100011000110'B),                          02050000
 SF_CCP  BIT(24) STATIC         /* CONDITIONAL PROC CNTL D3A7CA HEX */ 02060000
         INIT('110100111010011111001010'B),                          02070000
 SF_CFC  BIT(24) STATIC         /* CODED FONT CONTROL    D3A78A HEX */ 02080000
         INIT('110100111010011110001010'B),                          02090000
 SF_CFI  BIT(24) STATIC         /* CODED FONT INDEX      D38C8A HEX */ 02100000
         INIT('110100111000110010001010'B),                          02110000
 SF_CPC  BIT(24) STATIC         /* CODED PAGE CONTROL    D3A787 HEX */ 02120000
         INIT('110100111010011110000111'B),                          02130000
 SF_CPD  BIT(24) STATIC         /* CODED PAGE DESCRIPTOR D3A687 HEX */ 02140000
         INIT('110100111010011010000111'B),                          02150000
```

```
SF_CPI  BIT(24) STATIC         /* CODED PAGE INDEX      D3AC87 HEX */ 02160000
        INIT('110100111010110010000111'B),                             02170000
SF_CTC  BIT(24) STATIC         /* COMPOSED TEXT CONTROL D3A79B HEX */ 02180000
        INIT('110100111010011110011011'B),                             02190000
SF_CTD  BIT(24) STATIC         /* COMPOSED TEXT DESCR.  D3A69B HEX */ 02200000
        INIT('110100111010011010011011'B),                             02210000
SF_CTX  BIT(24) STATIC         /* COMPOSED TEXT DATA    D3EE9B HEX */ 02220000
        INIT('110100111110111010011011'B),                             02230000
SF_DXD  BIT(24) STATIC         /* DATA MAP SUBCASE DESC D3A6E3 HEX */ 02240000
        INIT('110100111010011011100011'B),                             02250000
SF_EAG  BIT(24) STATIC         /* END ACTIVE ENVIR GRP  D3A9C9 HEX */ 02260000
        INIT('110100111010100111001001'B),                             02270000
SF_EBC  BIT(24) STATIC         /* END BAR CODE OBJECT   D3A9EB HEX */ 02280037
        INIT('110100111010100111101011'B),                             02290037
SF_ECF  BIT(24) STATIC         /* END CODED FONT        D3A98A HEX */ 02300000
        INIT('110100111010100110001010'B),                             02310000
SF_ECP  BIT(24) STATIC         /* END CODED PAGE        D3A987 HEX */ 02320000
        INIT('110100111010100110000111'B),                             02330000
SF_ECT  BIT(24) STATIC         /* END COMPOSED TEXT BLK D3A99B HEX */ 02340000
        INIT('110100111010100110011011'B),                             02350000
SF_EDG  BIT(24) STATIC         /* END DOC ENVIRN GROUP  D3A9C4 HEX */ 02360000
        INIT('110100111010100111000100'B),                             02370000
SF_EDM  BIT(24) STATIC         /* END DATA MAP          D3A9CA HEX */ 02380000
        INIT('110100111010100111001010'B),                             02390000
SF_EDT  BIT(24) STATIC         /* END OF DOCUMENT       D3A9A8 HEX */ 02400000
        INIT('110100111010100110101000'B),                             02410000
SF_EDX  BIT(24) STATIC         /* END DATA MAP XMIT SUB D3A9E3 HEX */ 02420000
        INIT('110100111010100111100011'B),                             02430000
SF_EFG  BIT(24) STATIC         /* END FORM ENVIRON GRP  D3A9C5 HEX */ 02440000
        INIT('110100111010100111000101'B),                             02450000
SF_EFM  BIT(24) STATIC         /* END FORM MAP          D3A9CD HEX */ 02460000
        INIT('110100111010100111001101'B),                             02470000
SF_EFN  BIT(24) STATIC         /* END FONT              D3A989 HEX */ 02480000
        INIT('110100111010100110001001'B),                             02490000
SF_EGO  BIT(24) STATIC         /* END GRAPHICS OBJECT   D3A9BB HEX */ 02500037
        INIT('110100111010100110111011'B),                             02510037
SF_EIM  BIT(24) STATIC         /* END IMAGE BLOCK       D3A97B HEX */ 02520037
        INIT('110100111010100101111011'B),                             02530037
SF_EMM  BIT(24) STATIC         /* END MEDIUM MAP        D3A9CC HEX */ 02540000
        INIT('110100111010100111001100'B),                             02550000
SF_EMO  BIT(24) STATIC         /* END MEDIUM OVERLAY    D3A9DF HEX */ 02560000
        INIT('110100111010100111011111'B),                             02570000
SF_EOG  BIT(24) STATIC         /* END OBJECT ENVIR GRP  D3A9C7 HEX */ 02580037
        INIT('110100111010100111000111'B),                             02590037
SF_EPG  BIT(24) STATIC         /* END PAGE              D3A9AF HEX */ 02600000
        INIT('110100111010100110101111'B),                             02610000
SF_EPM  BIT(24) STATIC         /* END PAGE MAP          D3A9CB HEX */ 02620000
        INIT('110100111010100111001011'B),                             02630000
SF_EPS  BIT(24) STATIC         /* END   PAGE SEGMENT    D3A95F HEX */ 02640000
        INIT('110100111010100101011111'B),                             02650000
SF_EPT  BIT(24) STATIC         /* END PRESENT. TEXT BLK D3A99B HEX */ 02660025
        INIT('110100111010100110011011'B),                             02670025
SF_ER   BIT(24) STATIC         /* END RESOURCE          D3A9CE HEX */ 02680000
        INIT('110100111010100111001110'B),                             02690000
SF_ERG  BIT(24) STATIC         /* END RESOURCE GROUP    D3A9C6 HEX */ 02700000
        INIT('110100111010100111000110'B),                             02710000
SF_FDS  BIT(24) STATIC         /* FIXED DATA SIZE       D3AAEC HEX */ 02720000
        INIT('110100111010101011101100'B),                             02730000
SF_FDX  BIT(24) STATIC         /* FIXED DATA TEXT       D3EEEC HEX */ 02740000
        INIT('110100111110111011101100'B),                             02750000
SF_FGD  BIT(24) STATIC         /* FORM ENVIRN GRP DESC  D3A6C5 HEX */ 02760000
        INIT('110100111010011011000101'B),                             02770000
SF_FNC  BIT(24) STATIC         /* FONT CONTROL          D3A789 HEX */ 02780000
        INIT('110100111010011110001001'B),                             02790000
SF_FND  BIT(24) STATIC         /* FONT DESCRIPTOR       D3A689 HEX */ 02800000
        INIT('110100111010011010001001'B),                             02810000
```

```
SF_FNG  BIT(24) STATIC         /* FONT PATTERNS         D3EE89 HEX */ 02820000
        INIT('110100111110111010001001'B),                             02830000
SF_FNI  BIT(24) STATIC         /* FONT INDEX            D38C89 HEX */ 02840000
        INIT('110100111000110010001001'B),                             02850000
SF_FNM  BIT(24) STATIC         /* FONT PATTERNS MAP     D3A289 HEX */ 02860000
        INIT('110100111010001010001001'B),                             02870000
SF_FNO  BIT(24) STATIC         /* FONT ORIENTATION      D3AE89 HEX */ 02880000
        INIT('110100111010111010001001'B),                             02890000
SF_FNP  BIT(24) STATIC         /* FONT POSITION         D3AC89 HEX */ 02900000
        INIT('110100111010110010001001'B),                             02910000
SF_GAD  BIT(24) STATIC         /* GRAPHICS DATA         D3EEBB HEX */ 02920037
        INIT('110100111110111010111011'B),                             02930037
SF_GDD  BIT(24) STATIC         /* GRAPHICS DATA DESCRIP D3A6BB HEX */ 02940037
        INIT('110100111010011010111011'B),                             02950037
SF_ICP  BIT(24) STATIC         /* IMAGE CELL POSITION   D3AC7B HEX */ 02960037
        INIT('110100111010110001111011'B),                             02970037
SF_IDD  BIT(24) STATIC         /* IMAGE DATA DESCR IO   D3A6FB HEX */ 02980037
        INIT('110100111010011011111011'B),                             02990037
SF_IDM  BIT(24) STATIC         /* INVOKE DATA MAP       D3ABCA HEX */ 03000037
        INIT('110100111010101111001010'B),                             03010037
SF_IID  BIT(24) STATIC         /* IMAGE INPUT DESCRIPT. D3A67B HEX */ 03020000
        INIT('110100111010011001111011'B),                             03030033
SF_IMM  BIT(24) STATIC         /* IMAGE MEDIUM MAP      D3ABCC HEX */ 03040000
        INIT('110100111010101111001100'B),                             03050000
SF_IOC  BIT(24) STATIC         /* IMAGE OUTPUT CONTROL  D3A77B HEX */ 03060000
        INIT('110100111010011101111011'B),                             03070000
SF_IPD  BIT(24) STATIC         /* IMAGE PICTURE DATA    D3EEFB HEX */ 03080037
        INIT('110100111110111011111011'B),                             03090037
SF_IPO  BIT(24) STATIC         /* INCLUDE PAGE OVERLAY  D3AFD8 HEX */ 03100037
        INIT('110100111010111111011000'B),                             03110037
SF_IPS  BIT(24) STATIC         /* INCLUDE PAGE SEGMENT  D3AF5F HEX */ 03120037
        INIT('110100111010111101011111'B),                             03130037
SF_IRD  BIT(24) STATIC         /* IMAGE RASTER DATA     D3EE7B HEX */ 03140000
        INIT('110100111110111001111011'B),                             03150000
SF_LNC  BIT(24) STATIC         /* LINE DESCRIPTOR COUNT D3AAE7 HEX */ 03160000
        INIT('110100111010101011100111'B),                             03170000
SF_LND  BIT(24) STATIC         /* LINE DESCRIPTOR       D3A6E7 HEX */ 03180000
        INIT('110100111010011011100111'B),                             03190000
SF_MBC  BIT(24) STATIC         /* MAP BAR CODE          D3ABEB HEX */ 03200037
        INIT('110100111010101111111011'B),                             03210037
SF_MCC  BIT(24) STATIC         /* MEDIUM COPY COUNT     D3A288 HEX */ 03220037
        INIT('110100111010001010001000'B),                             03230037
SF_MCF  BIT(24) STATIC         /* MAP CODED FONT FMT 1  D3B18A HEX */ 03240037
        INIT('110100111011000110001010'B),                             03250000
SF_MCF2 BIT(24) STATIC         /* MAP CODED FONT FMT 2  D3AB8A HEX */ 03260037
        INIT('110100111010101110001010'B),                             03270037
SF_MDD  BIT(24) STATIC         /* MEDIUM DESCRIPTOR     D3A688 HEX */ 03280000
        INIT('110100111010011010001000'B),                             03290000
SF_MGO  BIT(24) STATIC         /* MAP GRAPHIC OBJECT    D3ABBB HEX */ 03300037
        INIT('110100111010101110111011'B),                             03310037
SF_MIO  BIT(24) STATIC         /* MAP IO IMAGE OBJECT   D3ABFB HEX */ 03320037
        INIT('110100111010101111111011'B),                             03330037
SF_MMC  BIT(24) STATIC         /* MEDIUM MODIFICATION   D3A788 HEX */ 03340037
        INIT('110100111010011110001000'B),                             03350037
SF_MMO  BIT(24) STATIC         /* MAP MEDIUM OVERLAY    D3B1DF HEX */ 03360000
        INIT('110100111011000111011111'B),                             03370000
SF_MPO  BIT(24) STATIC         /* MAP PAGE OVERLAY      D3ABD8 HEX */ 03380037
        INIT('110100111010101111011000'B),                             03390037
SF_MPS  BIT(24) STATIC         /* MAP PAGE SEGMENT      D3B15F HEX */ 03400037
        INIT('110100111011000101011111'B),                             03410037
SF_MSU  BIT(24) STATIC         /* MAP SUPPRESSION       D3ABEA HEX */ 03420029
        INIT('110100111010101111101010'B),                             03430029
SF_NOP  BIT(24) STATIC         /* NO OPERATION          D3EEEE HEX */ 03440000
        INIT('110100111110111011101110'B),                             03450000
SF_OBD  BIT(24) STATIC         /* OBJECT AREA DESCRIPT  D3A66B HEX */ 03460037
        INIT('110100111010011001101011'B),                             03470037
```

```
 SF_OBP  BIT(24) STATIC         /* OBJECT AREA POSITION  D3AC6B HEX */ 03480037
         INIT('110100111010110001101011'B),                            03490037
 SF_PGD  BIT(24) STATIC         /* PAGE DESCRIPTOR       D3A6AF HEX */ 03500000
         INIT('110100111010011010101111'B),                            03510000
 SF_PGP  BIT(24) STATIC         /* PAGE POSITION         D3ACAF HEX */ 03520000
         INIT('110100111010110010101111'B),                            03530000
 SF_PTD  BIT(24) STATIC         /* PRESENT. TEXT DESCR.  D3A69B HEX */ 03540000
         INIT('110100111010011010011011'B),                            03550000
 SF_PTX  BIT(24) STATIC         /* PRESENTATION TEXT     D3EE9B HEX */ 03560028
         INIT('110100111110111010011011'B),                            03570028
/*--------------------------------------------------------------*/     03580000
/* COMPOSED TEXT CONTROL SEQUENCES.                             */     03590000
/* TEXT CONTROLS ARE DESCRIBED AS FOLLOWS:                      */     03600000
/* BYTE 1: LENGTH. ZERO REPRESENTS A VARIABLE LENGTH CONTROL.   */     03610000
/* BYTE 2: TEXT-CONTROL CODE, UNCHAINED. (EVEN VALUES)          */     03620000
/*         CHAINED CONTROL CODES ARE REPRESENTED BY THE LOW-ORDER*/    03630000
/*         BIT TURNED ON (ODD VALUES).                          */     03640000
/*--------------------------------------------------------------*/     03650000
 TC_AMB BIT(16) STATIC          /* ABSOLUTE MOVE BASELINE  04D2 HEX */ 03660000
         INIT('0000010011010010'B),                                    03670000
 TC_AMI BIT(16) STATIC          /* ABSOLUTE MOVE INLINE    04C6 HEX */ 03680000
         INIT('0000010011000110'B),                                    03690000
 TC_BLN BIT(16) STATIC          /* BEGIN LINE              02D8 HEX */ 03700000
         INIT('0000001011011000'B),                                    03710000
 TC_BSU BIT(16) STATIC          /* BEGIN SUPPRESSION       03F2 HEX */ 03720000
         INIT('0000001111110010'B),                                    03730000
 TC_CCTL BIT(16) STATIC         /* CHAINED CONTROL FLAG BIT   1 HEX */ 03740000
         INIT('0000000000000001'B),                                    03750000
 TC_DBR BIT(16) STATIC          /* DRAW BASELINE RULE      07E6 HEX */ 03760000
         INIT('0000011111100110'B),                                    03770000
 TC_DIR BIT(16) STATIC          /* DRAW INLINE RULE        07E4 HEX */ 03780000
         INIT('0000011111100100'B),                                    03790000
 TC_ESC BIT(16) STATIC          /* ESCAPE SEQUENCE         2BD3 HEX */ 03800000
         INIT('0010101111010011'B),                                    03810000
 TC_ESU BIT(16) STATIC          /* END SUPPRESSION         03F4 HEX */ 03820000
         INIT('0000001111110100'B),                                    03830000
 TC_NOP BIT(16) STATIC          /* NO OPERATION            00F8 HEX */ 03840000
         INIT('0000000011111000'B),                                    03850000
 TC_RMB BIT(16) STATIC          /* RELATIVE MOVE BASELINE  04D4 HEX */ 03860000
         INIT('0000010011010100'B),                                    03870000
 TC_RMI BIT(16) STATIC          /* RELATIVE MOVE INLINE    04C8 HEX */ 03880000
         INIT('0000010011001000'B),                                    03890000
 TC_RPS BIT(16) STATIC          /* REPEAT STRING           00EE HEX */ 03900000
         INIT('0000000011101110'B),                                    03910000
 TC_SBI BIT(16) STATIC          /* SET BASELINE INCREMENT  04D0 HEX */ 03920000
         INIT('0000010011010000'B),                                    03930000
 TC_SCFL BIT(16) STATIC         /* SET CODED FONT LOCAL    03F0 HEX */ 03940026
         INIT('0000001111110000'B),                                    03950000
 TC_SII BIT(16) STATIC          /* SET INTERCHAR INCREMENT 04C2 HEX */ 03960000
         INIT('0000010011000010'B),                                    03970000
 TC_SIM BIT(16) STATIC          /* SET INLINE MARGIN       04C0 HEX */ 03980000
         INIT('0000010011000000'B),                                    03990000
 TC_STC BIT(16) STATIC          /* SET TEXT COLOR          0574 HEX */ 04000031
         INIT('0000010101110100'B),                                    04010031
 TC_STO BIT(16) STATIC          /* SET TEXT ORIENTATION    06F6 HEX */ 04020031
         INIT('0000011011110110'B),                                    04030031
 TC_SVI BIT(16) STATIC          /* SET VAR SPC CHAR INCR   04C4 HEX */ 04040000
         INIT('0000010011000100'B),                                    04050000
 TC_TRN BIT(16) STATIC          /* TRANSPARENT DATA        00DA HEX */ 04060000
         INIT('0000000011011010'B);                                    04070000
%GOTO DONE;                                                            04080035
%SFIDPLI2: ;                                                           04090000
%DCL DTYP ENTRY;                                                       04100021
%DTYP: PROC(HEX) RETURNS(CHAR);                                        04110016
 DCL (HEX, STR) CHAR;                                                  04120019
 IF GEN = 'CHAR'                                                       04130016
```

```
  THEN                                                                  04140016
   STR='INIT(''''||HEX||''''X)';                                        04150020
  ELSE                                                                  04160016
   STR='INIT(''''||HEX||''''BX)';                                       04170020
 RETURN(STR);                                                           04180019
%END DTYP;                                                              04190012
%DECLARE TYP CHAR;                                                      04200012
%IF GEN = 'CHAR'                                                        04210001
% THEN %DO;                                                            04220002
%  TYP = 'CHAR(3)';                                                     04230012
% END;                                                                  04240000
% ELSE %DO;                                                            04250002
%  TYP = 'BIT(24)';                                                     04260012
%END;                                                                   04270000
 DECLARE                                                                04280000
/*----------------------------------------------------------------*/    04290000
/* LAST UPDATE ON 08 SEP 1989 AT 10:30:14 BY VEND730 VERSION 02 */     04300000
/* SYMBOLIC EQUATES FOR STRUCTURED FIELD IDENTIFIERS AND          */     04310000
/* COMPOSED-TEXT CONTROLS. SEE PSF DATA STREAM REFERENCE,         */     04320000
/* SH35-0073-03.                                                  */     04330000
/* CHANGED FOR PL/I VER. 2, WHICH ALLOWS HEX BIT STRINGS.         */     04340000
/* FOR PL/I VER. 1, USE SFIDPLI, ABOVE.                           */     04350000
/* BIT       HEX VALUES MAY BE GENERATED BY PRECEEDING THE        */     04360005
/* INCLUDE FOR SFIDEQU BY THE FOLLOWING:                          */     04370000
/* %DCL GEN; GEN='BIT';                                           */     04380005
/* CHARACTER HEX VALUES MAY BE GENERATED BY PRECEEDING THE        */     04390005
/* INCLUDE FOR SFIDEQU BY THE FOLLOWING:                          */     04400005
/* %DCL GEN; GEN='CHAR';                                          */     04410005
/*----------------------------------------------------------------*/    04420000
/* SYMBOLS FOR STRUCTURED FIELD IDENTIFIERS:                      */     04430000
 SF_BAG  TYP STATIC           /* BEGIN ACTIVE ENVIRON    */             04440022
         DTYP(D3A8C9),                                                  04450017
 SF_BBC  TYP STATIC           /* BEGIN BAR CODE OBJECT   */             04460037
         DTYP(D3A8EB),                                                  04470037
 SF_BCF  TYP STATIC           /* BEGIN CODED FONT        */             04480022
         DTYP(D3A88A),                                                  04490022
 SF_BCP  TYP STATIC           /* BEGIN CODE PAGE         */             04500022
         DTYP(D3A887),                                                  04510022
 SF_BCT  TYP STATIC           /* BEGIN COMPOSED-TEXT     */             04520022
         DTYP(D3A89B),                                                  04530022
 SF_BDA  TYP STATIC           /* BAR CODE DATA           */             04540037
         DTYP(D3EEEB),                                                  04550037
 SF_BDD  TYP STATIC           /* BAR CODE DATA DESCRIPTOR*/             04560037
         DTYP(D3A6EB),                                                  04570037
 SF_BDG  TYP STATIC           /* BEGIN DOC ENVIRON GROUP */             04580022
         DTYP(D3A8C4),                                                  04590022
 SF_BDM  TYP STATIC           /* BEGIN DATA MAP          */             04600022
         DTYP(D3A8CA),                                                  04610022
 SF_BDT  TYP STATIC           /* BEGIN DOCUMENT          */             04620022
         DTYP(D3A8A8),                                                  04630022
 SF_BDX  TYP STATIC    /* BEGIN DATA MAP XMIT SUBCASE    */             04640022
         DTYP(D3A8E3),                                                  04650022
 SF_BFG  TYP STATIC    /* BEGIN FORM ENVIRONMENT GRP     */             04660022
         DTYP(D3A8C5),                                                  04670022
 SF_BFM  TYP STATIC    /* BEGIN FORM MAP                 */             04680022
         DTYP(D3A8CD),                                                  04690022
 SF_BFN  TYP STATIC    /* BEGIN FONT                     */             04700022
         DTYP(D3A889),                                                  04710022
 SF_BGR  TYP STATIC           /* BEGIN GRAPHICS OBJECT   */             04720037
         DTYP(D3A88B),                                                  04730037
 SF_BIM  TYP STATIC           /* BEGIN IMAGE BLOCK       */             04740037
         DTYP(D3A87B),                                                  04750037
 SF_BIMO TYP STATIC           /* BEGIN IMAGE OBJECT IO   */             04760037
         DTYP(D3A8FB),                                                  04770037
 SF_BMM  TYP STATIC           /* BEGIN MEDIUM MAP        */             04780022
         DTYP(D3A8CC),                                                  04790022
```

```
SF_BMO  TYP STATIC          /* BEGIN MEDIUM OVERLAY    */          04800022
        DTYP(D3A8DF),                                              04810022
SF_BOG  TYP STATIC          /* BEGIN OBJECT ENVIR GROUP*/          04820037
        DTYP(D3A8C7),                                              04830037
SF_BPG  TYP STATIC          /* BEGIN PAGE              */          04840037
        DTYP(D3A8AF),                                              04850037
SF_BPM  TYP STATIC          /* BEGIN PAGE MAP          */          04860022
        DTYP(D3A8CB),                                              04870022
SF_BPS  TYP STATIC          /* BEGIN PAGE SEGMENT      */          04880022
        DTYP(D3A85F),                                              04890022
SF_BPT  TYP STATIC       /* BEGIN PRESENTATION TEXT    */          04900022
        DTYP(D3A89B),                                              04910022
SF_BR   TYP STATIC          /* BEGIN RESOURCE          */          04920022
        DTYP(D3A8CE),                                              04930022
SF_BRG  TYP STATIC          /* BEGIN RESOURCE GROUP    */          04940022
        DTYP(D3A8C6),                                              04950022
SF_CCP  TYP STATIC          /* CONDITIONAL PROC CNT    */          04960022
        DTYP(D3A7CA),                                              04970022
SF_CFC  TYP STATIC          /* CODED FONT CONTROL      */          04980022
        DTYP(D3A78A),                                              04990022
SF_CFI  TYP STATIC          /* CODED FONT INDEX        */          05000022
        DTYP(D38C8A),                                              05010022
SF_CPC  TYP STATIC          /* CODED PAGE CONTROL      */          05020022
        DTYP(D3A787),                                              05030022
SF_CPD  TYP STATIC          /* CODED PAGE DESCRIPTO    */          05040022
        DTYP(D3A687),                                              05050022
SF_CPI  TYP STATIC          /* CODED PAGE INDEX        */          05060022
        DTYP(D3AC87),                                              05070022
SF_CTC  TYP STATIC          /* COMPOSED TEXT CONTRO    */          05080022
        DTYP(D3A79B),                                              05090022
SF_CTD  TYP STATIC          /* COMPOSED TEXT DESCR.    */          05100022
        DTYP(D3A69B),                                              05110022
SF_CTX  TYP STATIC          /* COMPOSED TEXT DATA      */          05120022
        DTYP(D3EE9B),                                              05130022
SF_DXD  TYP STATIC          /* DATA MAP SUBCASE DES    */          05140022
        DTYP(D3A6E3),                                              05150022
SF_EAG  TYP STATIC          /* END ACTIVE ENVIR GRP    */          05160022
        DTYP(D3A9C9),                                              05170022
SF_EBC  TYP STATIC          /* END BAR CODE OBJECT     */          05180037
        DTYP(D3A9EB),                                              05190037
SF_ECF  TYP STATIC          /* END CODED FONT          */          05200022
        DTYP(D3A98A),                                              05210022
SF_ECP  TYP STATIC          /* END CODED PAGE          */          05220022
        DTYP(D3A987),                                              05230022
SF_ECT  TYP STATIC          /* END COMPOSED TEXT BL    */          05240022
        DTYP(D3A99B),                                              05250022
SF_EDG  TYP STATIC          /* END DOC ENVIRN GROUP    */          05260022
        DTYP(D3A9C4),                                              05270022
SF_EDM  TYP STATIC          /* END DATA MAP            */          05280022
        DTYP(D3A9CA),                                              05290022
SF_EDT  TYP STATIC          /* END OF DOCUMENT         */          05300022
        DTYP(D3A9A8),                                              05310022
SF_EDX  TYP STATIC          /* END DATA MAP XMIT SU    */          05320022
        DTYP(D3A9E3),                                              05330022
SF_EFG  TYP STATIC          /* END FORM ENVIRON GRP    */          05340022
        DTYP(D3A9C5),                                              05350022
SF_EFM  TYP STATIC          /* END FORM MAP            */          05360022
        DTYP(D3A9CD),                                              05370022
SF_EFN  TYP STATIC          /* END FONT                */          05380022
        DTYP(D3A989),                                              05390022
SF_EGO  TYP STATIC          /* END GRAPHICS OBJECT     */          05400037
        DTYP(D3A9BB),                                              05410037
SF_EIM  TYP STATIC          /* END IMAGE BLOCK         */          05420037
        DTYP(D3A97B),                                              05430037
SF_EIMO TYP STATIC          /* END IMAGE BLOCK         */          05440037
        DTYP(D3A9FB),                                              05450037
```

```
SF_EMM  TYP STATIC          /* END MEDIUM MAP          */          05460022
        DTYP(D3A9CC),                                             05470022
SF_EMO  TYP STATIC          /* END MEDIUM OVERLAY      */          05480022
        DTYP(D3A9DF),                                             05490022
SF_EOG  TYP STATIC          /* END OBJECT ENVIRON GROUP*/          05500037
        DTYP(D3A9C7),                                             05510037
SF_EPG  TYP STATIC          /* END PAGE                */          05520022
        DTYP(D3A9AF),                                             05530022
SF_EPM  TYP STATIC          /* END PAGE MAP            */          05540022
        DTYP(D3A9CB),                                             05550022
SF_EPS  TYP STATIC          /* END     PAGE SEGMENT    */          05560022
        DTYP(D3A95F),                                             05570022
SF_EPT  TYP STATIC          /* END PRESENTATION TEXT   */          05580025
        DTYP(D3A99B),                                             05590025
SF_ER   TYP STATIC          /* END RESOURCE            */          05600022
        DTYP(D3A9CE),                                             05610022
SF_ERG  TYP STATIC          /* END RESOURCE GROUP      */          05620022
        DTYP(D3A9C6),                                             05630022
SF_FDS  TYP STATIC          /* FIXED DATA SIZE         */          05640022
        DTYP(D3AAEC),                                             05650022
SF_FDX  TYP STATIC          /* FIXED DATA TEXT         */          05660022
        DTYP(D3EEEC),                                             05670022
SF_FGD  TYP STATIC          /* FORM ENVIRN GRP DESC    */          05680022
        DTYP(D3A6C5),                                             05690022
SF_FNC  TYP STATIC          /* FONT CONTROL            */          05700022
        DTYP(D3A789),                                             05710022
SF_FND  TYP STATIC          /* FONT DESCRIPTOR         */          05720022
        DTYP(D3A689),                                             05730022
SF_FNG  TYP STATIC          /* FONT PATTERNS           */          05740022
        DTYP(D3EE89),                                             05750022
SF_FNI  TYP STATIC          /* FONT INDEX              */          05760022
        DTYP(D38C89),                                             05770022
SF_FNM  TYP STATIC          /* FONT PATTERNS MAP       */          05780022
        DTYP(D3A289),                                             05790022
SF_FNO  TYP STATIC          /* FONT ORIENTATION        */          05800022
        DTYP(D3AE89),                                             05810022
SF_FNP  TYP STATIC          /* FONT POSITION           */          05820022
        DTYP(D3AC89),                                             05830022
SF_GAD  TYP STATIC          /* GRAPHICS DATA           */          05840037
        DTYP(D3EEBB),                                             05850037
SF_GDD  TYP STATIC          /* GRAPHICS DATA DESCRIPT. */          05860037
        DTYP(D3A6BB),                                             05870037
SF_ICP  TYP STATIC          /* IMAGE CELL POSITION     */          05880037
        DTYP(D3AC7B),                                             05890037
SF_IDD  TYP STATIC          /* IMAGE DATA DESCRIPT. IO */          05900037
        DTYP(D3A6FB),                                             05910037
SF_IDM  TYP STATIC          /* INVOKE DATA MAP         */          05920037
        DTYP(D3ABCA),                                             05930037
SF_IID  TYP STATIC          /* IMAGE INPUT DESCRIPT    */          05940022
        DTYP(D3A67B),                                             05950022
SF_IMM  TYP STATIC          /* IMAGE MEDIUM MAP        */          05960022
        DTYP(D3ABCC),                                             05970022
SF_IOC  TYP STATIC          /* IMAGE OUTPUT CONTROL    */          05980022
        DTYP(D3A77B),                                             05990022
SF_IPD  TYP STATIC          /* IMAGE PICTURE DATA      */          06000037
        DTYP(D3EEFB),                                             06010037
SF_IPO  TYP STATIC          /* INCLUDE PAGE OVERLAY    */          06020037
        DTYP(D3AFD8),                                             06030037
SF_IPS  TYP STATIC          /* INCLUDE PAGE SEGMENT    */          06040037
        DTYP(D3AF5F),                                             06050037
SF_IRD  TYP STATIC          /* IMAGE RASTER DATA       */          06060022
        DTYP(D3EE7B),                                             06070022
SF_LNC  TYP STATIC          /* LINE DESCRIPTOR COUN    */          06080022
        DTYP(D3AAE7),                                             06090022
SF_LND  TYP STATIC          /* LINE DESCRIPTOR         */          06100022
        DTYP(D3A6E7),                                             06110022
```

```
 SF_MBC  TYP STATIC          /* MAP BAR CODE           */          06120037
        DTYP(D3ABEB),                                            06130037
 SF_MCC  TYP STATIC          /* MEDIUM COPY COUNT      */          06140037
        DTYP(D3A288),                                            06150037
 SF_MCF  TYP STATIC          /* MAP CODED FONT FORMAT 1 */          06160037
        DTYP(D3B18A),                                            06170022
 SF_MCF2 TYP STATIC          /* MAP CODED FONT FORMAT 2 */          06180037
        DTYP(D3AB8A),                                            06190037
 SF_MDD  TYP STATIC          /* MEDIUM DESCRIPTOR      */          06200022
        DTYP(D3A688),                                            06210022
 SF_MGO  TYP STATIC          /* MAP GRAPHIC OBJECT     */          06220037
        DTYP(D3ABBB),                                            06230037
 SF_MIO  TYP STATIC          /* MAP IO IMAGE OBJECT    */          06240037
        DTYP(D3ABFB),                                            06250037
 SF_MMC  TYP STATIC          /* MEDIUM MODIFICATION    */          06260022
        DTYP(D3A788),                                            06270022
 SF_MMO  TYP STATIC          /* MAP MEDIUM OVERLAY     */          06280022
        DTYP(D3B1DF),                                            06290022
 SF_MPO  TYP STATIC          /* MAP PAGE OVERLAY       */          06300037
        DTYP(D3ABD8),                                            06310037
 SF_MPS  TYP STATIC          /* MAP PAGE SEGMENT       */          06320037
        DTYP(D3B15F),                                            06330037
 SF_MSU  TYP STATIC          /* MAP SUPPRESSION        */          06340022
        DTYP(D3ABEA),                                            06350029
 SF_NOP  TYP STATIC          /* NO OPERATION           */          06360022
        DTYP(D3EEEE),                                            06370022
 SF_OBD  TYP STATIC          /* OBJECT AREA DESCRIPTOR  */          06380037
        DTYP(D3A66B),                                            06390037
 SF_OBP  TYP STATIC          /* OBJECT AREA POSITION   */          06400037
        DTYP(D3AC6B),                                            06410037
 SF_PGD  TYP STATIC          /* PAGE DESCRIPTOR        */          06420022
        DTYP(D3A6AF),                                            06430022
 SF_PGP  TYP STATIC          /* PAGE POSITION          */          06440022
        DTYP(D3ACAF),                                            06450022
 SF_PTD  TYP STATIC          /* PRESENT. TEXT DESCR.   */          06460022
        DTYP(D3A69B),                                            06470022
 SF_PTX  TYP STATIC          /* PRESENTATION TEXT DATA  */          06480028
        DTYP(D3EE9B),                                            06490028
/*------------------------------------------------------------------*/ 06500022
/* COMPOSED TEXT CONTROL SEQUENCES.                                 */ 06510022
/* TEXT CONTROLS ARE DESCRIBED AS FOLLOWS:                          */ 06520022
/* BYTE 1: LENGTH. ZERO REPRESENTS A VARIABLE LENGTH CODE.          */ 06530022
/* BYTE 2: TEXT-CONTROL CODE, UNCHAINED. (EVEN VALUES)              */ 06540022
/*         CHAINED CONTROL CODES ARE REPRESENTED BY THE HIGH-ORDER  */ 06550022
/*         BIT TURNED ON (ODD VALUES).                              */ 06560022
/*------------------------------------------------------------------*/ 06570022
%IF GEN = 'CHAR'                                                    06580022
% THEN %DO;                                                        06590022
%  TYP = 'CHAR(2)';                                                06600022
% END;                                                             06610022
% ELSE %DO;                                                        06620022
%  TYP = 'BIT(16)';                                                06630022
%END;                                                             06640022
 TC_AMB TYP STATIC           /* ABSOLUTE MOVE BASELI    */          06650022
        DTYP(04D2),                                              06660022
 TC_AMI TYP STATIC           /* ABSOLUTE MOVE INLINE    */          06670022
        DTYP(04C6),                                              06680022
 TC_BLN TYP STATIC           /* BEGIN LINE             */          06690022
        DTYP(02D8),                                              06700022
 TC_BSU TYP STATIC           /* BEGIN SUPPRESSION      */          06710022
        DTYP(03F2),                                              06720022
 TC_CCTL TYP STATIC          /* CHAINED CONTROL FLAG    */          06730022
        DTYP(0001),                                              06740022
 TC_DBR TYP STATIC           /* DRAW BASELINE RULE     */          06750022
        DTYP(07E6),                                              06760022
 TC_DIR TYP STATIC           /* DRAW INLINE RULE       */          06770022
```

```
        DTYP(07E4),                                                 06780022
 TC_ESC TYP STATIC          /* ESCAPE SEQUENCE        */           06790022
        DTYP(2BD3),                                                 06800022
 TC_ESU TYP STATIC          /* END SUPPRESSION        */           06810022
        DTYP(03F4),                                                 06820022
 TC_NOP TYP STATIC          /* NO OPERATION           */           06830022
        DTYP(00F8),                                                 06840022
 TC_RMB TYP STATIC          /* RELATIVE MOVE BASELI   */           06850022
        DTYP(04D4),                                                 06860022
 TC_RMI TYP STATIC          /* RELATIVE MOVE INLINE   */           06870022
        DTYP(04C8),                                                 06880022
 TC_RPS TYP STATIC          /* REPEAT STRING          */           06890022
        DTYP(00EE),                                                 06900022
 TC_SBI TYP STATIC          /* SET BASELINE INCREME   */           06910022
        DTYP(04D0),                                                 06920022
 TC_SCFL TYP STATIC         /* SET CODED FONT LOCAL   */           06930026
        DTYP(03F0),                                                 06940022
 TC_SII TYP STATIC          /* SET INTERCHAR INCREM   */           06950022
        DTYP(04C2),                                                 06960022
 TC_SIM TYP STATIC          /* SET INLINE MARGIN      */           06970022
        DTYP(04C0),                                                 06980022
 TC_STC TYP STATIC          /* SET TEXT COLOR         */           06990031
        DTYP(0574),                                                 07000031
 TC_STO TYP STATIC          /* SET TEXT ORIENTATION   */           07010022
        DTYP(06F6),                                                 07020022
 TC_SVI TYP STATIC          /* SET VAR SPC CHAR INC   */           07030022
        DTYP(04C4),                                                 07040022
 TC_TRN TYP STATIC          /* TRANSPARENT DATA       */           07050022
        DTYP(00DA);                                                 07060022
%DEACTIVATE GEN, TYP, DTYP;                                         07070022
%DONE: ;                                                            07080035
```

Appendix B. VM AFP Sample Programs

This appendix documents the exits and utility programs that we used on the VM platform to print our test cases.

All of the coding documented in this chapter is presented as *sample coding only*.

Be sure that you have read the information in “Special Notices” on page ix.

The following table serves as an index to the various routines.

Name	Language	Description	Page
AFPDSFIX	REXX	Splits a resource sent as binary into records to make them usable by PSF/VM.	155
CRTRSC	REXX	Processes AFP resource objects shipped from OS/400 to make them usable by PSF/VM.	158

B.1 AFPDSFIX routine for VM

This program fixes an AFP resource that has been uploaded as binary. When uploaded as binary, the record structure of a resource is lost, this routine splits the resource into records. After the procedure, the resource is usable in the VM system.

B.1.1 REXX Coding

```
/* REXX EXEC*/
/*                    */
 /* The routine examines the input file for AFPDS structured field   */
 /* records that may have been reformatted during a transmission     */
 /* from OS/2.  During such a transmission, the structured fields    */
 /* are treated as a stream of data and the original records are     */
 /* lost.  The transmitted file may have many structured fields in   */
 /* a single record, or an individual structured field may span      */
 /* multiple records.                                                */
 /*                                                                  */
 /* When properly formed AFPDS structured fields are detected they   */
 /* are written to the output as 1 structured field per output       */
 /* record.  Data not occuring within a structured field is written  */
 /* out according to the following rules:                            */
 /*                                                                  */
 /*    1. If no structured field introducer is present in the        */
 /*       record, the entire record is written unaltered.            */
 /*    2. If a structured field begins within the record, the data   */
 /*       preceding that structured field is written as a record.    */
 /*    3. Data falling between two valid structured fields is        */
 /*       written as a record.                                       */
 /*    4. Data following a valid structured field is written as a    */
 /*       record, with one exception.  If the structured field is    */
 /*       IPO, IPS, IMM, or IDM and the remainder of the record in   */
 /*       which it is found is blank, it is assumed to be a          */
 /*       valid structured field control record imbedded in a fixed  */
 /*       length record data file.  The trailing blanks are stripped */
 /*       and ignored.                                               */
 /*                                                                  */
 /* This logic has the following effects:                            */
 /*                                                                  */
```

© Copyright IBM Corp. 1994

```
 /*    1. Files with no structured field content are transcribed     */
 /*       verbatim.                                                  */
 /*                                                                  */
 /*    2. Files containing only structured fields are written out    */
 /*       with 1 structured field per record.                        */
 /*                                                                  */
 /*    3. Files with a mixture of structured field records and other */
 /*       data may or may not be reconstructed accurately.  Since    */
 /*       information about the original record lengths has been     */
 /*       lost, only AFPDS records can be accurately reconstructed.  */
 /*       If non-AFPDS data are isolated to their own records, the   */
 /*       reconstruction should be accurate.                         */
 /*                                                                  */
 /*------------------------------------------------------------------*/
 Parse upper arg fns
 parse var fns fid fn fm fid2 fn2 fm2 .
 state fid fn fm
if rc¬=0 then do
 Say 'file not found'
 exit
 end
infile = fid fn fm
outfile = fid2 fn2 fm2
/*Prime nextrec buffer*/ nextrec = readrec()
/*Clear currec        */ currec  = ""
/*Main execution loop */
/*                    */ Do Forever
/*Get an output rec   */    outrec = get_outrec()
/*If end of file, quit*/    if outrec = "*EOF*" then leave
/*Write output rec    */    "EXECIO 1 DISKW" outfile "(VAR OUTREC"
/*End main loop       */ end
/*                    */ ndit:
/*Close output file   */    "FINIS" outfile
/*Close input file    */    "FINIS" infile
/*Scram               */    return 0
 /*                                                                  */
 /*------------------------------------------------------------------*/
 /*                    Get Next Output Record                        */
 /* This routine isolates the next record to be written to the       */
 /* output file. CURREC contains the current data record, NEXTREC    */
 /* contains the next record from the input file.  When CURREC is    */
 /* fully processed, NEXTREC is moved to CURREC and a new record     */
 /* is read into NEXTREC from the input file.  The logic isolates    */
 /* the next output record using the rules described earler.  The    */
 /* isolated record is returned to the caller and stripped from      */
 /* CURREC.                                                          */
 /*------------------------------------------------------------------*/
 /*                                                                  */
/*                   */ Get_Outrec: procedure expose currec nextrec infile
/*If currec empty,    */    if length(currec) < 1 then do
/* move in nextrec and*/       currec = nextrec
/* read next record   */       nextrec = readrec()
/*                    */    end
/*Hit eof, quit       */    if currec = "*EOF*" then return currec
/*Start at position 0 */    candidate_pos = 0
/*Isolation loop      */    Do forever
/*Look for "!"        */       candidate_pos = pos("!",currec,candidate_pos+1)
/*If none             */       if candidate_pos = 0 then do
/* return whole record*/          outrec = currec
/* to caller          */          currec = ""
/*                    */          return outrec
/*                    */       end
/*Found a "!"         */       else do
/*Check for strfld rec*/          strlen = ver_strfld(currec||nextrec)
/*If not valid,       */          if strlen = -99 then do
/* bump position and  */             candidate_pos = candidate_pos + 1
```

```
/* look for next "!"  */            iterate
/*                    */          end
/*Found good strfld   */
/*Must loop to read   */
/* all data in strfld */
/*                    */
/*Get whole str field */          do while strlen > length(currec)
/* oops              */             if nextrec = "*EOF*" then do
/* bad str fld at end */                say "Found incomplete structured field record at",
/*                    */                    "end of input file"
/* save what we have  */                outrec = currec
/* blank currec       */                currec = ""
/* return incompl rec */                return outrec
/*                    */             end
/* append nextrec     */             currec = currec||nextrec
/* get another rec    */             nextrec = readrec()
/*end loop            */          end
/*Isolate strfld rec  */          strfld = substr(currec,1,strlen)
/*Update currec       */          currec = substr(currec,strlen+1)
/*Get strfld mneumonic*/          mneumonic = substr(strfld,4,3)
/*Str=IPO IPS IMM IDM */        chkstrng =
                                 x2c('d3afd840d3af5f40d3abcc40d3abca')
/*If rec one of these */          if 0 < wordpos(mneumonic,chkstrng),
/* and remainder blank*/           & strip(currec) = ""
/* assume fixed len   */          then currec = ""
/* str fld rec        */
/*Return to caller    */          return strfld
/*                    */       end
 /*                                                                  */
 /*------------------------------------------------------------------*/
 /*                    Read a Physical Record                        */
 /* This routine reads the next physical record from the input file  */
 /* and returns it to the caller.                                    */
 /*------------------------------------------------------------------*/
 /*                                                                  */
Readrec: procedure expose infile
/*                    */    "EXECIO 1 DISKR" infile "(VAR DISKREC"
/*                    */    if rc > 0 then return "*EOF*"
/*                    */    return diskrec
 /*                                                                  */
 /*------------------------------------------------------------------*/
 /*                   Verify a Structured Field                      */
 /* This routine verifys that the data string passed as an argument  */
 /* conforms to the rules for a valid AFPDS structured field.  If    */
 /* the argument appears to be a structured field, the length of     */
 /* structured field is retrieved from the AFPDS length field,       */
 /* incremented by 1 to account for the "!" byte and returned to     */
 /* the caller. If the argument doesn't verify, -99 is returned.     */
 /*------------------------------------------------------------------*/
 /*                                                                  */
Ver_Strfld: procedure
/*                    */    parse arg 1 cc 2 len 4 flag1 5 flag2 6 .
/*1st char x'5A'?     */    if cc    <> "!" then return -99
/*4th char x'D3'?     */    if flag1 <> "L" then return -99
/*5th char in list?   */    if 0 <>
                            verify(flag2,x2c('aaabacaeafa2a6a7a8a9b1b6ee8c')
                            then return -99
/*return strfld length*/    return 1+c2d(len)
```

B.2 OS/400 Resource Converter for VM

When the OS/400 SNDNETSPLF command is used to ship AFP print resource objects from OS/400 to VM, the object is not in a format acceptable to PSF/VM. The CRTRSC EXEC will read the resource from the VM virtual reader and convert the object into a format acceptable to PSF/VM.

B.2.1 REXX Coding

```
/* This 'CRTRSC' program creates an overlay          */
/* or a page segment from AFP U/400                  */
/* generated spooled file to VM                      */

trace 'Off'
address 'COMMAND'
parse upper source . . $fn . . $syn .
parse upper arg spid fn ft fm . '(' opts
if spid = '' | spid = '?' then signal tell

call init
trace value trace_opts
call read
call finish
exit

INIT:
  cpcmd = 'EXECIO 0 CP (STRING'
  call set_opts
  call check_spid
  call set_fileid

  tempfile = $fn '$$TEMP$$' fm
  call erase tempfile

  'EXECIO 1 CP (STRING QUERY VIRTUAL OOC'
  parse pull . . . rdr_class rdr_cont rdr_hold . rdr_ready .
  if rdr_ready <> 'READY' then call abort 36,'Reader OOC not ready.'
  cpcmd 'SPOOL OOC CLASS * NOCONT HOLD'
  cpcmd 'ORDER RDR' spid
  return

SET_OPTS:
  opt_errs = 0
  valid_opts = 'HOLD MSG NOMSG PURGE TRACE'
  parse value '0 1 Off' with opt.purge opt.msg trace_opts
  do forever
    parse var opts opt opts
    if opt = '' then leave
    opt1 = deabbrev(opt,valid_opts)
    if opt1 = '' then call opt_err 'Unrecognized option '''opt'''.'
    else select
      when opt1 = 'HOLD' then opt.purge = 0
      when opt1 = 'MSG' then opt.msg = 1
      when opt1 = 'NOMSG' then opt.msg = 0
      when opt1 = 'PURGE' then opt.purge = 1
      when opt1 = 'TRACE' then do
        parse var opts trace_opts opts
        end /* when */
      end /* else select */
  end /* do forever */
  if opt_errs > 0 then
    call abort opt_errs,'Option errors found.  Nothing done.'
  return

OPT_ERR:
```

```
  call warn arg(1)
  opt_errs = opt_errs + 1
  return

CHECK_SPID:
  if datatype(spid) <> 'NUM' then
    call abort 16,'Invalid spoolid '''spid'''.'
  if (trunc(spid)<>spid) | ((0+spid)<0) | ((0+spid)>9999) then
    call abort 16,'Invalid spoolid '''spid'''.'
  spid = right('000'||spid,4)
  'EXECIO * CP (STRING QUERY RDR' spid
  parse pull hdr
  if subword(hdr,3) = 'DOES NOT EXIST' then
    call abort 28,'Spoolid' spid 'does not exist.'
  parse pull . . . spf_type . . spf_hold .
  if spf_type <> 'PRT' then
    call abort 28,'Spoolid' spid 'is a' spf_type 'file, not PRT.'
  if spf_hold <> 'NONE' then do
     cpcmd 'CHANGE RDR' spid 'NOHOLD'
     call abort (rc%1000),'Can''t change HOLD status for spoolid' spid'.'
     end
  return

SET_FILEID:
  if (fn = '') | (find('= . *',fn) > 0) then fn = 'NONE'
  if (ft = '') | (find('= . *',ft) > 0) then ft = 'NONE'
  if fm <> '' then do
    'MAKEBUF'
    'LISTFILE $ $' fm '(LIFO'
    lrc = rc
    'DROPBUF'
    if lrc = 24 then
      call abort lrc,'Invalid filemode' fm'.  Nothing done.'
    if lrc = 36 then
      call abort lrc,'Disk' left(fm,1) 'not accessed.  Nothing done.'
    if ¬rw_disk(fm) then
      call abort 36,'Disk' left(fm,1) 'is read-only.  Nothing done.'
    end /* if */
  else fm = get_rw_disk()
  if verify(fn ft fm,'%*(=','Match') > 0 then
    call abort 20,'Invalid character in file id '''fn ft fm'''.'
  'STATE' fn ft fm
  if rc = 20 then exit rc
  return

READ:
  dowrite = 0
  sprecs = 0
  do forever
    parse value diag(14,'RNSB','00C') with code 2 . 9 buffer
    if code <> 0 then leave
    parse var buffer . 13 sprecnum +4 data
    sprecs = sprecs + c2d(sprecnum)
    do c2d(sprecnum)
      parse var data cc 2 . 5 flag 6 . 7 len 9 . 11 next 13 .
      select
        when flag = '70'x then do
          if dowrite = 1 then do
            'EXECIO 1 DISKW' tempfile '(VAR CC'
            call abort rc,'Unexpected error -- see file '''tempfile'''.'
            end
          data = substr(data,9)
          end /* when */
        when flag = '60'x then do
          if dowrite = 0 then sprecs = sprecs - 1
          line = cc || substr(data,13,c2d(len))
```

```
          if substr(line,4,3) = 'D3A8DF'x |,
             substr(line,4,3) = 'D3A85F'x then do
            if fn = 'NONE' then fn = substr(line,10,8)
            if ft = 'NONE' then
              if substr(line,6,1) = 'DF'x then
                ft = 'OVLY38PP'
              else
                ft = 'PSEG38PP'
            dowrite = 1
            end

          if dowrite = 1 then do
            recdata = cc || substr(data,13,c2d(len))
            'EXECIO 1 DISKW' tempfile '(VAR RECDATA'
            end
          data = substr(data,c2d(next)+1)
          if substr(line,4,3) = 'D3A9DF'x |,
             substr(line,4,3) = 'D3A95F'x then do
            sprecs = sprecs + 1
            dowrite = 0
            end
          end /* when */
        otherwise
          call abort 99,'Unrecognized CCW flags '''c2x(flag)'''.  ',
            'Execution halted.'
      end /* select */
    end /* do n */
  end /* do forever */
  return

FINISH:
  'FINIS' tempfile
  call erase fn ft fm
 'RENAME' tempfile fn ft fm
  if rc <> 0 then
    call warn 'Return code' rc 'from RENAME command. ',
      'See file '''tempfile'''.'
  cpcmd 'CLOSE OOC HOLD'
  cpcmd 'SPOOL OOC CLASS' rdr_class rdr_cont rdr_hold
  if opt.purge then cpcmd 'PURGE RDR' spid
  else if spf_hold <> 'NONE' then cpcmd 'CHANGE RDR' spid 'HOLD'
  if opt.msg then
    call warn 'File' fn ft fm 'created with' sprecs 'records.'
  return

RW_DISK:  procedure
  parse arg disk
  'QUERY DISK' left(disk,1) '(LIFO'
  parse pull . 16 acc_mode .
  if length(acc_mode) <= 3 then parse pull . /* scrap header line */
  return acc_mode = 'R/W'

GET_RW_DISK:
  'MAKEBUF'
  'QUERY DISK R/W (STACK FIFO'
  parse pull x .
  if x <> 'LABEL' then
    call abort 36, 'No Read/Write disk available.  Can''t continue.'
  parse pull . 12 x .
  'DROPBUF'
  return x

ERASE: procedure
  parse arg fn ft fm .
  'STATE' fn ft fm
  if rc = 0 then 'ERASE' fn ft fm
```

```
  return

DEABBREV: procedure
/* If 'first' is a prefix of some word in 'rest', then returns that word;
   else returns null string */
  parse upper arg first, rest
  if strip(first) = '' then return ''
  n = pos(' 'first,' 'rest)
  if n > 0 then return word(substr(rest,n),1)
  else return ''

ABORT:  procedure
  if arg(1) = 0 then return
  call warn arg(2)
  exit arg(1)

WARN:  procedure
  parse upper source . . . . . $syn .
  say $syn': ' arg(1)
  return

TELL:
'VMFCLEAR'
if $syn <> $fn then do
  say 'You have invoked' $fn $ft $fm 'under the synonym' $syn'.'
  say ''
  end
say 'Syntax: ' $fn 'spoolid <fn <ft <fm>>> <( options <)>>'
say ''
say 'Options:  HOLD -- Leave spooled file in reader'
say '          PURGE -- Delete spooled file after processing'
say '          MSG -- Give informational message after processing'
say '          NOMSG -- Forego message'
say ''
say 'Defaults:  fn -- overlay name from BMO structured field'
say '                 page segment name from BPS structured field'
say '           ft -- OVLY38PP (overlay)'
say '                 PSEG38PP (page segment)'
say '           fm -- first available Read/Write disk'
say '           Options -- HOLD MSG'
say ''
say 'If spooled file ''spoolid'' is in one''s reader and is of type'
say 'PRT,' $fn 'will extract the overlay or page segment from the'
say 'spooled file and save it into a file.'
exit 100
```

Appendix C. VSE AFP Sample Programs

This appendix documents the exits and utility programs that we used on the VSE platform to print our test cases.

All of the coding documented in this chapter is presented as *sample coding only*.

Be sure that you have read the information in “Special Notices” on page ix.

The following table serves as an index to the various routines.

Name	Language	Description	Page
AFPPUNCH	Assembler	Punches an AFP resource from the VSE library, includes JCL to initiate a job in the MVS system.	163
AFPPUNC2	Assembler	Punches an AFP resource from the VSE library, no JCL included, intended to be used by an EXEC in the VM system.	166
RESMAKE	Assembler	Creates an AFP resource in an MVS library from the VSE punched output	168
RESIN	Assembler	Copies AFP resources inline in front of the print file.	169
RESVM	REXX	Creates an AFP resource onto a VM CMS disk from an input file sent from the VSE system.	174
RESTAPE	Assembler	Copies an AFP resource to a tape file with variable length records.	175
RESAS4	Assembler	Copies an AFP resource to a tape file with fixed length records.	176
OS2PUNCH	C	Creates a VSE job from a resource in the OS/2 system	178
RESLINK	Assembler	Creates a link-edit job to link a resource to the VSE library	180
VSEPCH	Assembler	Creates a punch file from a resource in the VSE system.	184
VSE2OS2	C	Creates a resource into the OS/2 system using the input from a VSE system.	186

C.1 Program to Punch an AFP Resource for MVS

This program can be used in connection with C.3, “Program to Create a Resource from VSE Punch Output” on page 168 to extract an AFP resource from a VSE library, create a an input file for the MVS program, and create the necessary JCL statements to invoke the resource creating program in the MVS system. By setting the PDEST parameters in the POWER JECL properly, the resulting file is sent automatically to the MVS system.

C.1.1 IBM S/370 Assembler Coding for VSE

```
// JOB AFPPUNCH PUNCH A RESOURCE WITH JCL FOR MVS
// OPTION LINK,NOALIGN
// ASSGN SYSLST,FEE
// ASSGN SYSPCH,PUNCH
// EXEC ASSEMBLY,SIZE=512K
RESPUNCH CSECT
         BALR 12,0           ESTABLISH
         USING *,12          ADDRESSABILITY
         LA    13,SAVEA      LOAD ADDRESS OF SAVE AREA
```

© Copyright IBM Corp. 1994

```
         TM    0(1),X'80'    IS THERE A PARAMETER RECORD?
         BNO   NOPARM        NO, THEN QUIT
         L     2,0(1)        LOAD ADDRESS OF PARAMETER AREA
         LA    1,2(,2)       LOAD ADDRESS OF PARAMETER DATA
         LH    2,0(2)        LOAD LENGTH OF PARAMETER DATA
         SH    2,=H'1'       DECREASE LENGTH BY 1
         EX    2,PARMMOVE    MOVE THE PARAMETER DATA
         CLI   PARMAREA,X'40' IS THERE A NAME IN PARAMETER AREA
         BE    NOPARM         NO THEN QUIT
         MVC   FDEFNAME(8),PARMAREA  MOVE THE NAME
         LA    1,FDEFNAME           LOAD THE ADDRESS OF THE NAME
         CDLOAD (1),RETPNF=YES  ISSUE THE LOAD FOR THE PHASE
         LTR   15,15            WAS IT SUCCESSFUL?
         BNZ   NOPARM           NO, THEN QUIT
         ST    0,MODADDR        REQUESTED FORMDEF LOADED
         ST    1,MODENTRY       STORE REGISTERS
         ST    14,MODLENGT      ON THE TIME OF RETURN
         MVC   FDEFNAMX+1(8),FDEFNAME MOVE THE NAME TO TEMP FIELD
         LA    1,FDEFNAMX+8     THE ADDRESS OF 8TH LETTER OF THE NAME
         LA    2,7              SET COUNTER
CLI40    DS    0H
         CLI   0(1),X'40'       IS IT A BLANK
         BNE   LASTCHAR         NO, THEN THIS WAS THE LAST CHARACTER
         S     1,=F'1'          YES, GO TEST THE PREVIOUS CHAR
         BCT   2,CLI40
LASTCHAR DS    0H               THIS IS THE LAST NON-BLANK
         MVI   1(1),C')'        INSERT A RIGHT PARANTHESIS
         MVC   NAMEXXXX(10),FDEFNAMX  MOVE THE NAME TO THE OUTPUT JCL
         OPEN  PUNCH            OPEN THE PUNCH
         LA    2,RECAREA1       LOAD THE ADDRESS OF THE FIRST RECAREA
CLIRC1FF DS    0H
         CLI   0(2),X'FF'       IS THAT THE END
         BE    ENDREC1          YES
         MVC   IO1,0(2)         MOVE TO IOAREA
         PUT   PUNCH            PUNCH IT
         LA    2,81(,2)         NEXT RECORD
         B     CLIRC1FF         GO TEST OFR THE LAST
ENDREC1  DS    0H               LAST RECORD IN THE TABLE
         L     3,MODADDR        LOAD PHASE ADDRESS IN GETVIS
         L     5,MODLENGT      LOAD PHASE LENGTH
TESTL    DS    0H
         LTR   5,5              LENGTH ALREADY = 0
         BZ    END              YES, THEN FINISH
         MVC   HLENGTH,0(3)     MOVE RECORD LENGTH
         LH    4,HLENGTH        LOAD INTO REG 4
         SH    4,=H'4'          SUBSTRACT THE LENGTH OF THE PREFIX
         LA    2,4(3)           LOAD THE ADDRESS OF THE 5A RECORD
         BAL   10,PUTPUNCH      PUNCH IT ONTO THE CARDS
         AH    3,HLENGTH        ADD LENGTH TO THE ADDRESS
         SH    5,HLENGTH        DECREASE REMAINING LENGTH
         B     TESTL            GO BACK TO TEST
END      DS    0H               END OF RESOURCE
         LA    2,RECAREA2       LOAD ADDRESS OF THE SECOND RECORD AREA
CLIRC2FF DS    0H
         CLI   0(2),X'FF'       IS IT THE LAST RECORD?
         BE    ENDREC2          YES, THEN GO TO THE END
         MVC   IO1,0(2)         MOVE RECORD TO IOAREA
         PUT   PUNCH            PUNCH IT
         LA    2,81(,2)         GO TO THE NEXT RECORD
```

```
         B     CLIRC2FF         GO TO TEST FOR THE LAST RECORD
ENDREC2  DS    0H               LAST RECORD
         CLOSE PUNCH            SO THE JOB IS DONE
NOPARM   DS    0H
         EOJ
PUTPUNCH DS    0H               PUNCH THE RECORDS OF THE MODULE
         ST    4,PR4SAVE        STORE R4
         ST    2,PR2SAVE        STORE R2
         MVC   IO1(81),=CL81' ' CLEAR OUTPUT
         MVI   IO1+1,X'5A'      FIRST CHAR 5A INDICATES A NEW RECORD
PRLTR44  DS    0H
         LTR   4,4              LENGTH POSITIVE
         BNP   PRRET            NO, RETURN
         C     4,=F'60'         LENGTH LESS THAN 60?
         BNH   PUTREST          GO PUNCH THE LAST CARD
         MVC   PUTWORKA(60),0(2)     MOVE 60 CHARACTERS FROM MODULE
         PUT   PUNCH            PUNCH THE CARD
         MVC   IO1(81),=CL81' ' CLEAR OUTPUT
         LA    2,60(,2)         INCREASE POINTER BY 60
         S     4,=F'60'         DECREASE REMAINING LENGTH BY 60
         B     PRLTR44          GO TEST LENGTH
PUTREST  DS    0H               LAST CARD
         S     4,=F'1'          DECREASE LENGTH BY 1
         EX    4,PUTMOVE        MOVE THE REMAINING BYTES
         PUT   PUNCH            PUNCH
PRRET    DS    0H               RETURN
         L     4,PR4SAVE        LOAD THE
         L     2,PR2SAVE        SAVED REGISTERS
         BR    10               BRANCH BACK TO CALLING ROUTINE
PUTMOVE  MVC   PUTWORKA(1),0(2)
PARMMOVE MVC   PARMAREA(1),0(1)
IO1      DC    CL81' '
PUTWORKA EQU   IO1+2
SAVEA    DS    9D
FDEFNAME DS    D
PDEFNAME DS    D
PR4SAVE  DS    F
PR2SAVE  DS    F
MODADDR  DS    F
MODLENGT DS    F
MODENTRY DS    F
HLENGTH  DS    H
PARMAREA DS    0CL100
         DC    100C' '
         LTORG
PUNCH    DTFDI DEVADDR=SYSPCH,IOAREA1=IO1,RECSIZE=80
FDEFNAMX DC    C'('
         DC    CL8' '
         DC    C')'
RECAREA1 DC    CL81' //PRTMIKKO JOB (00000),''MARKKULA'',CLASS=A,'
         DC    CL81' //         MSGCLASS=X'
         DC    CL81' //*'
         DC    CL81' /*ROUTE PRINT WTSCSL2'
         DC    CL81' //O1    OUTPUT  DEFAULT=YES'
         DC    CL81' //STEP1 EXEC PGM=RESMAKE '
         DC    CL81' //STEPLIB DD DSN=PRTMIKK.MTM.LIB,DISP=SHR'
NR       DC    CL81' //SYSUT2 DD DISP=OLD,DSN=PRTMIKK.FONTLIB'
NAMEXXXX EQU   NR+41
         DC    CL81' //SYSPRINT DD SYSOUT=X'
```

```
         DC    CL81' //SYSIN    DD DUMMY'
         DC    CL81' //SYSUT1 DD *'
         DC    X'FF'
RECAREA2 DC    CL81' //'
         DC    X'FF'
         END   RESPUNCH
/*
// EXEC LNKEDT
// LIBDEF PHASE,SEARCH=PRD2.AFP
// EXEC ,PARM='T1000437'
/*
/&
```

C.2 Program to Punch an AFP Resource for VM

This program creates an input file for the REXX EXEC C.5, "Program to Create a Resource in VM" on page 174. The only difference with C.1, "Program to Punch an AFP Resource for MVS" on page 163 is that this program does not create JCL around the resource, as the output of this program is used for a REXX EXEC. It loads an AFP resource from the VSE library, splits it into 60 character length records, and puts the records to the POWER/VSE punch queue. If the output record starts a new AFP record, the output record is prefixed with a X'5A' character, otherwise the record will be prefixed by a blank character.

From the punch queue the file can be sent to a VM system by specifying the PDEST parameter in the POWER JECL JOB statement.

C.2.1 IBM S/370 Assembler Coding for VSE

```
// JOB AFPPUNC2 PUNCH A RESOURCE FOR VM
// OPTION LINK,NOALIGN
// ASSGN SYSLST,FEE
// ASSGN SYSPCH,PUNCH
// EXEC ASSEMBLY,SIZE=512K
RESPUNCH CSECT
         BALR 12,0           ESTABLISH
         USING *,12          ADDRESSABILITY
         LA    13,SAVEA      LOAD ADDRESS OF SAVE AREA
         TM    0(1),X'80'    IS THERE A PARAMETER RECORD?
         BNO   NOPARM        NO, THEN QUIT
         L     2,0(1)        LOAD ADDRESS OF PARAMETER AREA
         LA    1,2(,2)       LOAD ADDRESS OF PARAMETER DATA
         LH    2,0(2)        LOAD LENGTH OF PARAMETER DATA
         SH    2,=H'1'       DECREASE LENGTH BY 1
         EX    2,PARMMOVE    MOVE THE PARAMETER DATA
         CLI   PARMAREA,X'40' IS THERE A NAME IN PARAMETER AREA
         BE    NOPARM         NO THEN QUIT
         MVC   FDEFNAME(8),PARMAREA  MOVE THE NAME
         LA    1,FDEFNAME           LOAD THE ADDRESS OF THE NAME
         CDLOAD (1),RETPNF=YES  ISSUE THE LOAD FOR THE PHASE
         LTR   15,15            WAS IT SUCCESSFUL?
         BNZ   NOPARM           NO, THEN QUIT
         ST    0,MODADDR        REQUESTED FORMDEF LOADED
         ST    1,MODENTRY       STORE REGISTERS
         ST    14,MODLENGT      ON THE TIME OF RETURN
         OPEN  PUNCH            OPEN THE PUNCH
         L     3,MODADDR        LOAD PHASE ADDRESS IN GETVIS
```

```
         L     5,MODLENGT      LOAD PHASE LENGTH
TESTL    DS    0H
         LTR   5,5              LENGTH ALREADY = 0
         BZ    END              YES, THEN FINISH
         MVC   HLENGTH,0(3)     MOVE RECORD LENGTH
         LH    4,HLENGTH        LOAD INTO REG 4
         SH    4,=H'4'          SUBSTRACT THE LENGTH OF THE PREFIX
         LA    2,4(3)           LOAD THE ADDRESS OF THE 5A RECORD
         BAL   10,PUTPUNCH      PUNCH IT ONTO THE CARDS
         AH    3,HLENGTH        ADD LENGTH TO THE ADDRESS
         SH    5,HLENGTH        DECREASE REMAINING LENGTH
         B     TESTL            GO BACK TO TEST
END      DS    0H               END OF RESOURCE
         CLOSE PUNCH            SO THE JOB IS DONE
NOPARM   DS    0H
         EOJ
PUTPUNCH DS    0H               PUNCH THE RECORDS OF THE MODULE
         ST    4,PR4SAVE        STORE R4
         ST    2,PR2SAVE        STORE R2
         MVC   IO1(81),=CL81' ' CLEAR OUTPUT
         MVI   IO1+1,X'5A'      FIRST CHAR 5A INDICATES A NEW RECORD
PRLTR44  DS    0H
         LTR   4,4              LENGTH POSITIVE
         BNP   PRRET            NO, RETURN
         C     4,=F'60'         LENGTH LESS THAN 60?
         BNH   PUTREST          GO PUNCH THE LAST CARD
         MVC   PUTWORKA(60),0(2)     MOVE 60 CHARACTERS FROM MODULE
         PUT   PUNCH            PUNCH THE CARD
         MVC   IO1(81),=CL81' ' CLEAR OUTPUT
         LA    2,60(,2)         INCREASE POINTER BY 60
         S     4,=F'60'         DECREASE REMAINING LENGTH BY 60
         B     PRLTR44          GO TEST LENGTH
PUTREST  DS    0H               LAST CARD
         S     4,=F'1'          DECREASE LENGTH BY 1
         EX    4,PUTMOVE        MOVE THE REMAINING BYTES
         PUT   PUNCH            PUNCH
PRRET    DS    0H               RETURN
         L     4,PR4SAVE        LOAD THE
         L     2,PR2SAVE        SAVED REGISTERS
         BR    10               BRANCH BACK TO CALLING ROUTINE
PUTMOVE  MVC   PUTWORKA(1),0(2)
PARMMOVE MVC   PARMAREA(1),0(1)
IO1      DC    CL81' '
PUTWORKA EQU   IO1+2
SAVEA    DS    9D
FDEFNAME DS    D
PDEFNAME DS    D
PR4SAVE  DS    F
PR2SAVE  DS    F
MODADDR  DS    F
MODLENGT DS    F
MODENTRY DS    F
HLENGTH  DS    H
PARMAREA DS    0CL100
         DC    100C' '
         LTORG
PUNCH    DTFDI DEVADDR=SYSPCH,IOAREA1=IO1,RECSIZE=80
         END   RESPUNCH
/*
```

```
// EXEC LNKEDT
// LIBDEF PHASE,SEARCH=PRD2.AFP
// EXEC ,PARM='T1000437'
/*
/&
```

C.3 Program to Create a Resource from VSE Punch Output

This program is used for creating an AFP resource in the MVS system from the output created by C.1, "Program to Punch an AFP Resource for MVS" on page 163 in the VSE system.

This program has to be link-edited into an MVS library before invoking the program with a batch job from the VSE system.

C.3.1 IBM S/370 Assembler Coding for MVS

```
RESMAKE  CSECT
         PRINT NOGEN
         USING *,R15               ESTABLISH ADDRESSABILITY
         STM R14,R12,12(R13)       SAVE REGISTERS
         BALR  12,0                USE REG 12 AS BASE REGISTER
         DROP  15                  DROP USE OF REG 15
         USING *,12                AND USE REG 12
         ST    R1,PARMSAVE         ST REG1
         LA    R1,SAVE1            LOAD ADDRESS OF THE NEW SAVE
         ST    R1,8(R13)           AREA AND
         ST    R13,SAVE1+4         PUT IT INTO THE CHAIN
         LR    R13,R1
         OPEN  (GETDCB,INPUT)      OPEN INPUT
         OPEN  (PUTDCB,OUTPUT)     AND OUTPUT
         LA    R8,OUTREC+4         POINT TO THE 5TH BYTE OF OUTREC
READREC  DS   0H
         BAL  R10,GETREC           READ INPUT
         CLI  INREC,X'5A'          5A IDENTIFIES A NEW RECORD
         BNE  CONTINUE             NOT A NEW, THEN OLD CONTINUED
         CLI  FIRSTIND,X'FF'       NEW, IS IT FIRST
         BNE  FIRST                IT IS THE FIRST, SO SKIP
         BAL  R10,PUTREC           OLD RECORD OUTPUTTED
         LA   R8,OUTREC+4          POINT TO 5TH BYTE
FIRST    DS   0H
         MVI  FIRSTIND,X'FF'       SET FIRST OFF
CONTINUE DS   0H
         MVC  0(60,R8),INREC+1     MOVE DATA BYTES TO OUTPUT
         LA   R8,60(,R8)           INCREASE POINTER BY 60
         B    READREC              GET NEXT
ERRXIT   EQU   *                   THIS IS ENTERED AT EOF
         BAL  R10,PUTREC           THEN WE JUST PUT THE OLD
EXIT     DS   0H
         CLOSE (GETDCB)            CLOSE
         CLOSE (PUTDCB)            FILES
         L     R13,SAVE1+4         LOAD REGISTERS
         LM    R14,R12,12(R13)     FOR RETURNING
SETRCODE LA    R15,0               RETURN CODE IN REG 15
RCODE    EQU   SETRCODE+3
         BR    R14                 GET OUT
GETREC   DS    0H
         GET   GETDCB,INREC        GET INPUT RECORD
```

```
         BR    R10
PUTREC   DS    0H
         LH    R8,OUTREC+5          LOAD OUTPUT RECORD LENGTH
         AH    R8,=H'5'             ADD 5 TO INCLUDE HEADER
         STH   R8,OUTREC            STORE LENGTH IN FRONT
         MVC   OUTREC+2(2),=X'0000' CLEAR NEXT TWO BYTES
         PUT   PUTDCB,OUTREC        PUT THE OUTPUT RECORD
         BR    R10
SAVE1    DS    18F
R0       EQU   0
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9
R10      EQU   10
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
         LTORG
PUTDCB   DCB   MACRF=PM,DSORG=PS,RECFM=VBM,DDNAME=SYSUT2,              X
               LRECL=32756,BLKSIZE=32760
GETDCB   DCB   MACRF=GM,DSORG=PS,RECFM=FB,DDNAME=SYSUT1,EODAD=ERRXIT,  X
               LRECL=80,BLKSIZE=3120
         DS    0H
PARMSAVE DC    F'0'
FIRSTIND DC    X'00'
INREC    DS    CL80
OUTREC   DS    CL10000
         DS    CL10000
         DS    CL10000
         DS    CL10000
         END RESMAKE
```

C.4 Program to Punch an AFP Resource Inline

There are no tools in the VSE system to include resources inline in front of the print data set. This program is an example how to access the resources from the VSE libraries and print them as an inline resource group.

The program takes the names and types of the resources from parameter cards:

```
Columns
1       8 9      16
Type      Name
```

Type is FORMDEF for a form definition, PAGEDEF for a page definition, OVERLAY for an overlay, SEGMENT for a page segment, FONT for a coded font, CPAGE for a code page, and CHARSET for a character set.

C.4.1 IBM S/370 Assembler Coding for VSE

```
* $$ JOB JNM=MIKKOTST,DISP=D,CLASS=0,PRI=9
* $$ LST DISP=H,CLASS=X
* $$ LST DISP=D,CLASS=U,LST=02E,                                          C
* $$ DEST=(WTSCSL2,PR3825),PAGEDEF=DUMMY,FORMDEF=DUMMY
// JOB RESINCL  TESTING FOR INCLUDING AFP RESOURCES
// LIBDEF PHASE,CATALOG=PRD2.AFP
// OPTION CATAL,NOALIGN
 PHASE MMRESUPR,*
// ASSGN SYS010,02E
// ASSGN SYSLST,00E
// EXEC ASSEMBLY,SIZE=512K
RESIN1   CSECT
         BALR 12,0             ESTABLISH
         USING *,12            ADDRESSABILITY
         LA    13,SAVEA        SAVE AREA ADDRESS
         LA    6,RESTABLE      POINTER TO THE START OF TABLE
         OPEN  CARDIN          OPEN CARD READER (PARAMETERS)
CARDREAD DS    0H
         GET   CARDIN,CARD     GET A CARD
         MVC   0(16,6),CARD    MOVE INFORMATION INTO THE TABLE
         LA    6,16(,6)        INCREASE POINTER
         B     CARDREAD        BACK TO READ
CARDEOF  DS    0H
         MVI   0(6),X'FF'      MARK THE END OF THE TABLE
         OPEN  PRINTER         OPEN PRINTER FILE
         LH    4,HRECBRG       PUT
         PUT   PRINTER,BRG     BEGIN RESOURCE GROUP
         LA    6,RESTABLE      POINTER TO THE START OF THE TABLE
CLIEND   DS    0H
         CLI   0(6),X'FF'      AT END?
         BE    RESEND          YES, EXIT
         MVC   RESNAME(8),8(6) MOVE RESOURCE NAME
         LA    1,RESNAME       LOAD THE RESOURCE
         CDLOAD (1),RETPNF=YES FROM THE LIBRARY
         LTR   15,15           FOUND IT?
         BNZ   NEXTRES         NO, TO THE NEXT ENTRY
         ST    0,MODADDR       STORE MODULE ADDRESS
         ST    1,MODENTRY      STORE ENTRY POINT (NOT USED)
         ST    14,MODLENGT     STORE MODULE LENGTH
         CLC   =C'CPAGE',0(6)  A CODE PAGE?
         BE    INLINECP        YES
         CLC   =C'CHARSET',0(6) A CHARACTER SET?
         BE    INLINECS        YES
         CLC   =C'FORMDEF',0(6) A FORM DEFINITION?
         BE    INLINEFD        YES
         CLC   =C'PAGEDEF',0(6) A PAGE DEFINITION?
         BE    INLINEPD        YES
         CLC   =C'OVERLAY',0(6) AN OVERLAY?
         BE    INLINEOV        YES
         CLC   =C'SEGMENT',0(6) A PAGE SEGMENT?
         BE    INLINEPS        YES
         CLC   =C'FONT',0(6)   A CODED FONT
         BE    INLINEFN        YES
NEXTRES  DS    0H              NEXT ENTRY
         LA    6,16(,6)        INCREASE POINTER
         B     CLIEND          BACK TO TEST THE END
INLINEFD DS    0H              FORM DEFINITION
         LH    4,HRECFD        LOAD ADDRESS OF BR RECORD
```

```
         MVC   BRFDEF,8(6)      MOVE NAME
         PUT   PRINTER,BRFD     PUT THE BR RECORD
         B     PUNCHRES         GO TO PRINT THE RESOURCE
INLINEPD DS    OH               PAGE DEFINITION
         LH    4,HRECPD         LOAD ADDRESS OF BR RECORD
         MVC   BRPDEF,8(6)      NOVE NAME
         PUT   PRINTER,BRPD     PUT THE BR RECORD
         B     PUNCHRES         GO TO PRINT THE RESOURCE
INLINEOV DS    OH               OVERLAY
         LH    4,HRECOV         LOAD ADDRESS OF BR RECORD
         MVC   BROVER,8(6)      MOVE NAME
         PUT   PRINTER,BROV     PUT THE BR RECORD
         B     PUNCHRES         GO TO PRINT THE RESOURCE
INLINEPS DS    OH               PAGE SEGMENT
         LH    4,HRECPS         LOAD ADDRESS OF THE BR RECORD
         MVC   BRPSEG,8(6)      MOVE NAME
         PUT   PRINTER,BRPS     PUT THE BR RECORD
         B     PUNCHRES         GO TO PRINT THE RESOURCE
INLINEFN DS    OH               CODED FONT
         LH    4,HRECFN         LOAD ADDRESS OF THE BR RECORD
         MVC   BRFONT,8(6)      MOVE NAME
         PUT   PRINTER,BRFN     PUT THE BR RECORD
         B     PUNCHRES         GO TO PRINT THE RECORD
INLINECS DS    OH               CHARACTER SET
         LH    4,HRECCS         LOAD ADDRESS OF THE BR RECORD
         MVC   BRCSET,8(6)      MOVE NAME
         PUT   PRINTER,BRCS     PUT THE BR RECORD
         B     PUNCHRES         GO TO PRINT THE RESOURCE
INLINECP DS    OH               CODE PAGE
         LH    4,HRECCP         LOAD ADDRESS OF THE BR RECORD
         MVC   BRCPAG,8(6)      MOVE NAME
         PUT   PRINTER,BRCP     PUT THE BR RECORD
PUNCHRES DS    OH
         MVC   ERNAME(8),8(6)   MOVE NAME TO THE ER RECORD
         L     3,MODADDR        LOAD PHASE ADDRESS IN GETVIS
         L     5,MODLENGT       LOAD PHASE LENGTH
TESTL    DS    OH
         LTR   5,5              TEST FOR REMAINING PHASE LENGTH
         BZ    END              0, SO IT IS THE END
         MVC   HLENGTH,0(3)     MOVE LENGTH OF THE RECORD
         LH    4,HLENGTH        LOAD INTO REG 4
         SH    4,=H'4'          DECREASE BY 4 (RECORD LENGTH FIELD)
         LA    2,4(3)           LOAD ADDRESS OF THE 5A RECORD
         PUT   PRINTER,(2)      PUT THE RECORD
         AH    3,HLENGTH        MOVE POINTER TO THE NEXT RECORD
         SH    5,HLENGTH        DECREASE REMAINING LENGTH
         B     TESTL            GO TEST REMAINING LENGTH
END      DS    OH               END
         LH    4,HRECER         PUT
         PUT   PRINTER,ER       END RESOURCE
         B     NEXTRES
RESEND   DS    OH
         LH    4,HRECERG       PUT
         PUT   PRINTER,ERG     END RESOURCE GROUP
         CLOSE PRINTER
         EOJ
RECORD   DS    CL80
IN       DS    CL80
OUT      DS    CL80
```

```
DIN       DS    CL80
DOUT      DS    CL88
          LTORG
SAVEA     DS    9D
RESNAME   DS    D
MODADDR   DS    F
MODLENGT  DS    F
MODENTRY  DS    F
HLENGTH   DS    H
CARD      DS    CL80
*
BRG       DC    X'5A0010D3A8C6000000',C'RESGROUP'
BRGE      EQU   *
HRECBRG   DC    AL2(BRGE-BRG)
*
BRFD      DC    X'5A001AD3A8CE000000'
BRFDEF    DC    CL8'F1MIKKO1'
          DC    X'00000821FE0000000000'
BRFDE     EQU   *
HRECFD    DC    AL2(BRFDE-BRFD)
*
ER        DC    X'5A0010D3A9CE000000'
ERNAME    DC    CL8'F1MIKKO1'
ERE       EQU   *
HRECER    DC    AL2(ERE-ER)
*
BRPD      DC    X'5A001AD3A8CE000000'
BRPDEF    DC    CL8'P1MIKKO1'
          DC    X'00000821FD0000000000'
BRPDE     EQU   *
HRECPD    DC    AL2(BRPDE-BRPD)
*
BRPS      DC    X'5A001AD3A8CE000000'
BRPSEG    DC    CL8'S1MIKKO1'
          DC    X'00000821FB0000000000'
BRPSE     EQU   *
HRECPS    DC    AL2(BRPSE-BRPS)
*
BROV      DC    X'5A001AD3A8CE000000'
BROVER    DC    CL8'O1MIKKO1'
          DC    X'00000821FC0000000000'
BROVE     EQU   *
HRECOV    DC    AL2(BROVE-BROV)
*
BRFN      DC    X'5A001AD3A8CE000000'
BRFONT    DC    CL8'XOMIKKO1'
          DC    X'0000082142000000000'
BRFNE     EQU   *
HRECFN    DC    AL2(BRFNE-BRFN)
*
BRCS      DC    X'5A001AD3A8CE000000'
BRCSET    DC    CL8'COMIKKO1'
          DC    X'0000082140000000000'
BRCSE     EQU   *
HRECCS    DC    AL2(BRCSE-BRCS)
*
BRCP      DC    X'5A001AD3A8CE000000'
BRCPAG    DC    CL8'T1MIKKO1'
```

```
         DC    X'00000821410000000000'
BRCPE    EQU   *
HRECCP   DC    AL2(BRCPE-BRCP)
*
ERG      DC    X'5A0010D3A9C6000000',C' RESGROUP'
ERGE     EQU   *
HRECERG  DC    AL2(ERGE-ERG)
*
CARDIO1  DS    CL80
RESTABLE DS    CL320
IO1      DS    CL512
         DS    CL512
         DS    CL512
         DS    CL512
         DS    CL512
         DS    CL512
         DS    CL512
         DS    CL512
         DS    CL512
         DS    CL512
         DS    CL512
         DS    CL512
         DS    CL512
         DS    CL512
         DS    CL512
         DS    CL512
         DS    CL512
         DS    CL512
         DS    CL512
         DS    CL512
         DS    CL512
         DS    CL512
         DS    CL512
         DS    CL512
         DS    CL512
         DS    CL512
         DS    CL512
         DS    CL512
         DS    CL512
         DS    CL512
         DS    CL512
         DS    CL512
         DS    CL512
CARDIN   DTFCD DEVADDR=SYSIPT,IOAREA1=CARDIO1,WORKA=YES,TYPEFLE=INPUT, C
               EOFADDR=CARDEOF
PRINTER  DTFPR DEVADDR=SYS010,IOAREA1=IO1,BLKSIZE=512,RECSIZE=(4),     C
               RECFORM=UNDEF,CTLCHR=ASA,DEVICE=PRT1,WORKA=YES
         END   RESIN1
/*
// EXEC LNKEDT
// LIBDEF PHASE,SEARCH=PRD2.AFP
// EXEC
PAGEDEF P1OWN
FORMDEF F1OWN
/*
/&
* $$ EOJ
```

C.5 Program to Create a Resource in VM

The following EXEC creates a resource from the input created by C.2, “Program to Punch an AFP Resource for VM” on page 166.

The format of the input file is special for this EXEC.

To create a resource, it has to be received onto a CMS disk, then this EXEC can be used to create the resource.

C.5.1 REXX EXEC Coding for VM

```
/* REXX */
/* This routine creates a resource from the file sent from VSE */
/* The input file is created by a special program in VSE       */
/*                                                             */
/* To invoke this enter                                        */
/*   VSE2VM oldname ot om newname nt nm                        */
/*         where oldname is the name of the file from VSE      */
/*               ot      is the type of the file from VSE      */
/*               om      is the mode of the file from VSE      */
/*               newname is the name of the file to be created */
/*               nt      is the type of the file to be created */
/*               nm      is the mode of the file to be created */
/*                                                             */
/* This EXEC builds the variable length records from the input */
/* and then adjusts the length to the correct 5A-record length */
/*                                                             */
 Parse upper arg fns
 parse var fns fid fn fm fid2 fn2 fm2 .
 state fid fn fm
if rc¬=0 then do
 Say 'file not found'
 exit
 end
/*
fid2 = fid
fn2 = 'AFPR3820'
fm2 = '*'
*/
"ERASE" fid2 fn2 fm2
"EXECIO * DISKR " fid fn fm  " (FINIS STEM RIVI. "
COUNT = RIVI.0
TEMP = ""
DO I = 1 TO COUNT
 IF SUBSTR(RIVI.I,1,1)=X2C('5A') THEN DO
   IF LENGTH(TEMP)>0 THEN DO
    len5a=c2d(substr(temp,2,1))*256+c2d(substr(temp,3,1))+1
    temp = substr(temp,1,len5a)
    "EXECIO 1 DISKW " fid2 fn2 fm2 " (VAR TEMP "
    END
  TEMP=SUBSTR(RIVI.I,2,60)
  END
 ELSE DO
   TEMP=TEMP || SUBSTR(RIVI.I,2,60)
  END
 END
    len5a=c2d(substr(temp,2,1))*256+c2d(substr(temp,3,1))+1
```

```
    temp = substr(temp,1,len5a)
"EXECIO 1 DISKW " fid2 fn2 fm2 " (VAR TEMP "
QUEUE ""
"EXECIO * DISKW " fid2 fn2 fm2 " (FINIS "
```

C.6 Program to Create a Tape File for MVS or VM

The following program copies an AFP resource from the VSE system library onto a tape file for transferring to an MVS or a MVS system. The tape file has undefined length records.

In MVS, the file can be copied to an AFP library by using the IEBGENER utility program.

IN VM, the file can be copied by using the MOVEFILE command.

C.6.1 IBM S/370 Assembler Coding for VSE

```
* $$ JOB JNM=LIBTEST,CLASS=0,PRI=9,DISP=D
// JOB LISTEST  TESTING FOR IMBEDDING AFP RECORDS
// OPTION LINK,NOALIGN
// ASSGN SYSLST,FEE
// ASSGN SYSPCH,PUNCH
// EXEC ASSEMBLY,SIZE=512K
RESPUNCH CSECT
         BALR 12,0           ESTABLISH
         USING *,12          ADDRESSABILITY
         LA   13,SAVEA       LOAD ADDRESS OF SAVE AREA
         TM   0(1),X'80'     IS THERE A PARAMETER RECORD?
         BNO  NOPARM         NO, THEN QUIT
         L    2,0(1)         LOAD ADDRESS OF PARAMETER AREA
         LA   1,2(,2)        LOAD ADDRESS OF PARAMETER DATA
         LH   2,0(2)         LOAD LENGTH OF PARAMETER DATA
         SH   2,=H'1'        DECREASE LENGTH BY 1
         EX   2,PARMMOVE     MOVE THE PARAMETER DATA
         CLI  PARMAREA,X'40' IS THERE A NAME IN PARAMETER AREA
         BE   NOPARM          NO THEN QUIT
         MVC  FDEFNAME(8),PARMAREA  MOVE THE NAME
         LA   1,FDEFNAME           LOAD THE ADDRESS OF THE NAME
         CDLOAD (1),RETPNF=YES  ISSUE THE LOAD FOR THE PHASE
         LTR  15,15             WAS IT SUCCESSFUL?
         BNZ  NOPARM            NO, THEN QUIT
         ST   0,MODADDR         REQUESTED FORMDEF LOADED
         ST   1,MODENTRY        STORE REGISTERS
         ST   14,MODLENGT       ON THE TIME OF RETURN
         OPEN TAPEOUT           OPEN THE OUTPUT
         L    7,MODADDR         LOAD PHASE ADDRESS IN GETVIS
         L    8,MODLENGT       LOAD PHASE LENGTH
TESTL    DS   0H
         LTR  8,8               LENGTH ALREADY = 0
         BZ   END               YES, THEN FINISH
         MVC  HLENGTH,0(7)      MOVE RECORD LENGTH
         LH   3,HLENGTH         LOAD INTO REG 3
         SH   3,=H'4'           DECREASE BY 4 TO THE ACTUAL LENGTH
         LA   2,4(7)            LOAD THE ADDRESS OF THE 5A RECORD
         PUT  TAPEOUT,(2)       PUT THE RECORD ONTO THE TAPE
         AH   7,HLENGTH         ADD LENGTH TO THE ADDRESS
         SH   8,HLENGTH         DECREASE REMAINING LENGTH
```

```
         B     TESTL            GO BACK TO TEST
END      DS    0H               END OF RESOURCE
         CLOSE TAPEOUT          SO THE JOB IS DONE
NOPARM   DS    0H
         EOJ
PARMMOVE MVC   PARMAREA(1),0(1)
SAVEA    DS    9D
FDEFNAME DS    D
PDEFNAME DS    D
PR4SAVE  DS    F
PR2SAVE  DS    F
MODADDR  DS    F
MODLENGT DS    F
MODENTRY DS    F
HLENGTH  DS    H
PARMAREA DS    0CL100
         DC    100C' '
         LTORG
IO1      DS    CL16384
TAPEOUT  DTFMT DEVADDR=SYS011,IOAREA1=IO1,TYPEFLE=OUTPUT,BLKSIZE=16388,X
               RECFORM=UNDEF,FILABL=STD,RECSIZE=(3),WORKA=YES
         END   RESPUNCH
/*
// EXEC LNKEDT
// LIBDEF PHASE,SEARCH=PRD2.AFP
// ASSGN SYS011,181,00
// TLBL TAPEOUT,'VM/MVS FILE'
// EXEC ,PARM='T1000437'
/*
/&
* $$ EOJ
```

C.7 Program to Create a Tape File for AS/400

The following program copies an AFP resource from a VSE system library onto a tape file for transferring the resource to an AS/400 system. To facilitate the process on the AS/400 side, the tape file has fixed length records, in our case the size 16384 bytes, which should be enough for any resource.

On the AS/400, the tape file is copied to a physical file member with the CPYFRMTAP command, and then transformed to a resource with an appropriate CRT command.

C.7.1 IBM S/370 Assembler Coding for VSE

```
* $$ JOB JNM=LIBTEST,CLASS=0,PRI=9,DISP=D
// JOB LISTEST  TESTING FOR IMBEDDING AFP RECORDS
// OPTION LINK,NOALIGN
// ASSGN SYSLST,FEE
// ASSGN SYSPCH,PUNCH
// EXEC ASSEMBLY,SIZE=512K
RESPUNCH CSECT
         BALR 12,0           ESTABLISH
         USING *,12          ADDRESSABILITY
         LA    13,SAVEA      LOAD ADDRESS OF SAVE AREA
         TM    0(1),X'80'    IS THERE A PARAMETER RECORD?
         BNO   NOPARM        NO, THEN QUIT
         L     2,0(1)        LOAD ADDRESS OF PARAMETER AREA
```

```
         LA    1,2(,2)       LOAD ADDRESS OF PARAMETER DATA
         LH    2,0(2)        LOAD LENGTH OF PARAMETER DATA
         SH    2,=H'1'       DECREASE LENGTH BY 1
         EX    2,PARMMOVE    MOVE THE PARAMETER DATA
         CLI   PARMAREA,X'40' IS THERE A NAME IN PARAMETER AREA
         BE    NOPARM         NO THEN QUIT
         MVC   FDEFNAME(8),PARMAREA  MOVE THE NAME
         LA    1,FDEFNAME           LOAD THE ADDRESS OF THE NAME
         CDLOAD (1),RETPNF=YES  ISSUE THE LOAD FOR THE PHASE
         LTR   15,15            WAS IT SUCCESSFUL?
         BNZ   NOPARM           NO, THEN QUIT
         ST    0,MODADDR        REQUESTED FORMDEF LOADED
         ST    1,MODENTRY       STORE REGISTERS
         ST    14,MODLENGT      ON THE TIME OF RETURN
         OPEN  TAPEOUT          OPEN THE OUTPUT
         L     7,MODADDR        LOAD PHASE ADDRESS IN GETVIS
         L     8,MODLENGT      LOAD PHASE LENGTH
TESTL    DS    0H
         LTR   8,8              LENGTH ALREADY = 0
         BZ    END              YES, THEN FINISH
         MVC   HLENGTH,0(7)     MOVE RECORD LENGTH
         LH    3,HLENGTH        LOAD INTO REG 4
         SH    3,=H'4'          DECREASE BY FOUR TO GET ACTUAL LENGTH
         ICM   3,B'1000',=X'40' BLANK AS THE FILL CHARACTER
         LH    5,=H'16384'      SIZE OF OUT FIXED LENGTH RECORD
         LA    2,4(7)           LOAD THE ADDRESS OF THE 5A RECORD
         LA    4,IO1            REG 4 TO POINT TO IOAREA
         MVCL  4,2              MOVE THE RECORD
         PUT   TAPEOUT          PUNT THE RECORD ONTO THE TAPE
         AH    7,HLENGTH        ADD LENGTH TO THE ADDRESS
         SH    8,HLENGTH        DECREASE REMAINING LENGTH
         B     TESTL            GO BACK TO TEST
END      DS    0H               END OF RESOURCE
         CLOSE TAPEOUT          SO THE JOB IS DONE
NOPARM   DS    0H
         EOJ
PARMMOVE MVC   PARMAREA(1),0(1)
SAVEA    DS    9D
FDEFNAME DS    D
PDEFNAME DS    D
PR4SAVE  DS    F
PR2SAVE  DS    F
MODADDR  DS    F
MODLENGT DS    F
MODENTRY DS    F
HLENGTH  DS    H
PARMAREA DS    0CL100
         DC    100C' '
         LTORG
IO1      DS    CL16384
TAPEOUT  DTFMT DEVADDR=SYS011,IOAREA1=IO1,TYPEFLE=OUTPUT,BLKSIZE=16384,X
               RECFORM=FIXUNB,FILABL=STD
         END   RESPUNCH
/*
// EXEC LNKEDT
// LIBDEF PHASE,SEARCH=PRD2.AFP
// ASSGN SYS011,181,00
// TLBL TAPEOUT,'AS/400 FILE'
// EXEC ,PARM='T1000437'
```

```
/*
/&
* $$ EOJ
```

C.8 Program to Create a Job for VSE

This program takes the name of the input file (the resource to be dumped) and the name of the output file (a temporary file) where the dumped resource with JCL is written as an ASCII file. The third argument to the program is a parameter record including three eight-byte fields: library name, sublibrary name, and member name for the resource to be dumped and uploaded to the VSE system.

The output file includes POWER JECL and VSE JCL statements (in ASCII format) in front of and after the resource. The resource is dumped in 60 character records in hexadecimal ASCII format. This format was chosen to avoid the problems caused by the code conversion when uploading the file.

The output file can be uploaded to the VSE system using the Intelligent Workstation feature by entering the following command SEND outfile (FILE=RDR. This command puts the output file in the POWER reader queue.

C.8.1 C Coding for OS/2

```
#include <stdio.h>
#include <io.h>
#include <ctype.h>
#include <string.h>

 char hexchars[16]="0123456789ABCDEF";
 char jcbstr[21]="* $$ JOB JNM=OS22VSE\n";
 char jcbst2[21]="* $$ PUN DISP=I     \n";
 char jcbst3[21]="* $$ EOJ            \n";
 char jobstr[21]="// JOB LIBRTEST     \n";
 char jobstr2[38]="// LIBDEF PHASE,SEARCH=PRD2.AFP      \n";
 char jobstr3[53]=
   "// EXEC LIBRTEST,PARM='                         '     \n";
 char jobstr4[5]="/*  \n";
 char jobstr5[5]="/&  \n";
     FILE    *stream;
       FILE    *stream2;
       char    namebuf[15];
       char    namebuf2[15];
       char    namebuf3[26];
       char    buffer[62];
       char    *name;
       char    *name2;
       char    *name2;
       char    *name3;
       char ch;
       int     numread;
       int i;
       int i1;
       int i2;
       int imax;
main(argc,argv)
   int argc;
```

```
   char *argv[];
{
       /* Get a file if one was not specified as an argument   */
       if (argc > 1)
        name = argv[1];
       else {
 printf ("Enter output file name: ");
        name = gets(namebuf);
       }
 if (argc > 2)
        name2= argv[2];
       else {
        printf("Enter input file name: ");
        name2 = gets(namebuf2);
       }

 if (argc > 3)
        name3= argv[3];
       else {
        printf("Enter parameters: ");
        name3 = gets(namebuf3);
       }

       /* Open files in binary mode     */

       if ((stream = fopen(name,"w")) == NULL)
            return(1);
       if ((stream2 = fopen(name2,"rb")) == NULL)
        return (1);
   /* Move parameter field on the card */
   for (i=0;i<24;i++)
   jobstr3[23+i]=name3[i];
  /* write JECL and JCL records in front of the resource */
    numread=fwrite(jcbstr,1,21,stream);
    numread=fwrite(jcbst2,1,21,stream);
    numread=fwrite(jobstr,1,21,stream);
    numread=fwrite(jobstr2,1,38,stream);
    numread=fwrite(jobstr3,1,53,stream);
    i=0;
    imax=60;
 hachar:
        ch=getc(stream2);
    if (feof(stream2))
    {goto finish;}
/* change the char read to two hex chars */
     i1 = (unsigned char) ch/16;
     i2= (unsigned char) ch-16*i1;
     buffer[i]=hexchars[i1];
     buffer[i+1]=hexchars[i2];
     i+=2;
/* if record full, write it on the disk */
      if (i==imax)
{
buffer[imax]=0x0d;
buffer[imax+1]=0x0a;
    numread=fwrite(buffer,1,imax+2,stream);
    i=0;}
   goto hachar;
/* write the last record */
```

```
finish:
buffer[i]=0x0d;
buffer[i+1]=0x0a;
numread=fwrite(buffer,1,i+2,stream);
  /* write JECL and JCL records after the resource */
    numread=fwrite(jobstr4,1,5,stream);
    numread=fwrite(jobstr5,1,5,stream);
    numread=fwrite(jcbst3,1,21,stream);
      return (0);
}
```

C.9 Program to Create the Linkage Editor Job

This program runs in the VSE system. It is started by sending a reader file fom the OS/2 system to the VSE system. This program punches another job in POWER reader. The job created will link the resource in the appropriate resource library.

C.9.1 IBM S/370 Assembler Coding for VSE

```
* $$ JOB JNM=MIKKOLIB,DISP=D,CLASS=0,PRI=9
// JOB OS22VSER RESOURCE FROM OS/2 to VSE
// LIBDEF PHASE,CATALOG=PRD2.AFP
// ASSGN SYSLST,00E
// OPTION CATAL
 PHASE LIBRTEST,*
// EXEC ASSEMBLY,SIZE=256K
         PRINT GEN
TEST     CSECT
         BALR  12,0           ESTABLISH
         USING *,12           ADDRESSABILITY
         LA    13,SAVEA       LOAD SAVEA AREA ADDRESS
         TM    0(1),X'80'     IS THERE A PARAMETER RECORD?
         BNO   NOPARM         NO, THEN QUIT
         L     2,0(1)         LOAD ADDRESS OF PARAMETER AREA
         LA    1,2(,2)        LOAD ADDRESS OF PARAMETER DATA
         LH    2,0(2)         LOAD LENGTH OF PARAMETER DATA
         SH    2,=H'1'        DECREASE LENGTH BY 1
         EX    2,PARMMOVE     MOVE THE PARAMETER DATA
         CLI   PARMAREA,X'40' IS THERE A NAME IN PARAMETER AREA
         BE    NOPARM         NO THEN QUIT
         MVC   MEMBNAME(8),PARMAREA+16 SAVE MEMBER NAME
         MVC   LIBSLIB(8),PARMAREA     MOVE LIBRARY NAME
         LA    5,LIBSLIB               POINTER TO START OF LIBR.NAME
CLILIB40 DS    0H
         CLI   0(5),X'40'              BLANK?
         BE    FOUBLANK                YES, END OF LIBR.NAME
         LA    5,1(,5)                 NO, INCREASE POINTER
         B     CLILIB40                BACK TO CHECK
FOUBLANK DS    0H                      FOUND SPACE
         MVI   0(5),C'.'               INSERT A DOT
         MVC   1(8,5),PARMAREA+8       AND MOVE SUBLIBR.NAME
         OPEN  CARDS                   OPEN INPUT FILE
         LA    5,MODAREA               POINTER TO START OF RESOURCE
         SR    9,9                     CLEAR R9
GETCARD  DS    0H
         GET   CARDS,INCARD            GET A CARD
         LA    6,INCARD                POINTER TO START OF THE CARD
```

```
         LA    7,30                    MAX 30 BYTES (60 HEX DIGITS)
CLI40    DS    0H
         CLI   0(6),X'40'              A BLANK?
         BE    LOPPU                   YES, QUIT
         TR    0(1,6),TRTAUL           TRANSLATE FIRST DIGIT
         TR    1(1,6),TRTAUL           TRANSLATE SECOND DIGIT
         SR    8,8                     CLEAR R8
         SR    10,10                   CLEAR R10
         IC    8,0(6)                  INSERT FIRST DIGIT
         SLL   8,4                     SHIFT IT TO THE RIGHT PLACE
         IC    10,1(6)                 INSERT  SECOND DIGIT
         AR    8,10                    DIGITS TO THE SAME BYTE
         STC   8,0(5)                  STORE CHAARCTER
         LA    5,1(,5)                 INCREASE OUTPUT POINTER
         LA    9,1(,9)                 INCREASE NUMBER OF BYTES
         LA    6,2(,6)                 INCREASE INPUT POINTER
         BCT   7,CLI40                 DATA LEFT, BACK TO PROCESS
         C     9,MAXDATA               COMPARE IF AREA FULL
         BH    LOPPU2                  YES, DUMP
         B     GETCARD                 NO, GET ANOTHER CARD
LOPPU2   DS    0H                      AREA FULL, DUMP
         DUMP
LOPPU    DS    0H                      BLANKS FOUND IN INPUT
         CLOSE CARDS                   CLOSE INPUT
         ST    9,LEN                   STORE TOTAL LENGTH
         L     2,=A(MOD2AREA)          POINTER TO SECOND AREA
         LA    8,MODAREA               POINTER TO INPUT AREA
         LR    5,9                     MOVE LENGTH TO REG 5
CLI5A    DS    0H
         CLI   0(8),X'5A'              IS IT 5A-RECORD
         BNE   LOPPU3                  NO THEN WE ARE AT THE END
         MVC   HLEN(2),1(8)            MOVE LENGTH FIELD
         LH    6,HLEN                  LOAD IT TO REG 6
         LR    1,6                     AND REG 1
         A     6,=F'1'                 ADD ONE (5A-BYTE)
         STH   6,REC5AL                STORE 5A-RECORD LENGTH
         A     6,=F'4'                 ADD 4 BYTES FOR REC LENGTH
         STH   6,HLEN                  STORE TO LENGTH
         MVC   0(2,2),HLEN             MOVE TO MODAREA2
         MVC   2(3,2),=X'00005A'       ZEROS OF RECL AND 5A
         LA    2,5(,2)                 INCREASE OUTPUT POINTER
         LA    8,1(,8)                 INCREASE INPUT POINTER
LOOPCHAR DS    0H
         MVC   0(1,2),0(8)             MOVE FROM AREA TO AREA2
         LA    2,1(,2)                 INCREASE OUTPUT POINTER
         LA    8,1(,8)                 INCREASE INPUT POINTER
         BCT   1,LOOPCHAR              BACK, IF STILL BYTES
         L     1,FLEN                  INCREASE
         AH    1,HLEN                  THE TOTAL LENGTH
         ST    1,FLEN                  AND STORE BACK
         SH    5,REC5AL                DECREASE 5A-RECL FROM INPUT
         BP    CLI5A                   BACK IF STILL POSITIVE
LOPPU3   DS    0H
         MVC   LEN,FLEN                MOVE LENGTH OF RESULT
         OPEN  PUNCH           OPEN PUNCH FILE
         LA    2,RECAREA       LOAD ADDRESS OF THE JCL RECORD AREA
CLIRCAFF DS    0H
         CLI   0(2),X'FF'      IS IT THE LAST RECORD?
         BE    ENDREC1         YES, THEN GO TO THE END
```

```
         MVC   IO1,0(2)          MOVE RECORD TO IOAREA
         PUT   PUNCH             PUNCH IT
         LA    2,81(,2)          GO TO THE NEXT RECORD
         B     CLIRCAFF          GO TO TEST FOR THE LAST RECORD
ENDREC1  DS    0H                LAST RECORD
         MVC   PHASECD+8(8),MEMBNAME  MOVE MEMBER NAME TO PHASE CARD
         LA    1,PHASECD+8             POINTER TO THE START OF NAME
         L     2,8                     LENGTH MAX 8
CLI40NAM DS    0H
         CLI   0(1),X'40'              IS IT BLANK
         BE    LESST8                  YES
         LA    1,1(,1)                 NO, INCREASE POINTER
         BCT   2,CLI40NAM              AND BACK, IF STILL CHARS
LESST8   DS    0H
         MVC   0(L'PHASEX,1),PHASEX    MOVE TEXT AFTER PHASENAME
         MVC   IO1,PHASECD             MOVE TO IOAREA
         PUT   PUNCH                   AND PUNCH
         MVC   ESDRECA,INITADDR+1      MOVE INITADDR TO ESD CARD
         MVC   ESDRECL,LEN+1           MOVE LENTH TO ESD CARD
         MVC   ESDRECN,MEMBNAME        MOVE NAME TO ESD CARD
         MVC   IO1,ESDREC              MOVE TO IOAREA
         PUT   PUNCH                   AND PUNCH
         L     4,=A(MOD2AREA)          POINTER TO THE START OF MODULE
         L     5,LEN                   LENGTH OF THE MODULE
         MVC   ADDR,INITADDR           INITIALE ADDRESS
SHR5     DS    0H
         SH    5,=H'56'                SUBSTR 56 (BYTES/CARD)
         BNP   LASTREC                 IF NOT POSITIVE, LAST CARD
         MVC   TXTRECA,ADDR+1          MOVE CURRENT ADDR TO TXT CARD
         MVC   TXTRECL,=H'56'          MOCE LENGTH TO TXT CARD
         MVC   TXTRECD(56),0(4)        MOVE CONSTANT FIELD TO TXT CARD
         MVC   IO1,TXTREC              MOVE TO IOAREA
         PUT   PUNCH                   AND PUNCH
         L     1,ADDR                  INCREASE
         AH    1,=H'56'                ADDRESS
         ST    1,ADDR                  AND STORE
         MVC   TXTRECD,=CL56' '        CLEAR TXT CARD
         LA    4,56(,4)                INCREASE INPUT POINTER
         B     SHR5                    BACK TO COMPARE
LASTREC  DS    0H                      THE LAST CARD
         AH    5,=H'56'                ADD 56 TO GET LENGTH
         STH   5,HTEMP                 STORE IT
         MVC   TXTRECL,HTEMP           MOVE TO THE TXT CARD
         BCTR  5,0                     DECREASE BY 1 FOR EX
         EX    5,OUTMOVE               MOVE THE BYTES
         MVC   TXTRECA,ADDR+1          MOVE CURRENT ADDRESS
         MVC   IO1,TXTREC              MOVE TO IOAREA
         PUT   PUNCH                   AND PUNCH
         MVC   ENDRECA,INITADDR+1      MVC INIT.ADDR. TO END CARD
         MVC   IO1,ENDREC              MOVE TO IOAREA
         PUT   PUNCH                   AND PUNCH
         LA    2,RECAREA2        LOAD ADDRESS OF THE SECOND JCL AREA
CLIRC2FF DS    0H
         CLI   0(2),X'FF'        IS IT THE LAST RECORD?
         BE    ENDREC2           YES, THEN GO TO THE END
         MVC   IO1,0(2)          MOVE RECORD TO IOAREA
         PUT   PUNCH             PUNCH IT
         LA    2,81(,2)          GO TO THE NEXT RECORD
```

```
         B     CLIRC2FF         GO TO TEST FOR THE LAST RECORD
ENDREC2  DS    0H               LAST RECORD
NOPARM   DS    0H
         EOJ
PARMMOVE MVC   PARMAREA(1),0(1)
OUTMOVE  MVC   TXTRECD(0),0(4)
SAVEA    DS    9D
INITADDR DC    X'00027078'
ADDR     DC    F'0'
MAXDATA  DC    F'32000'   ADJUST TO MATCH THE AREA
HTEMP    DS    H
RECAREA  DC    CL81' // JOB LINK'
         DC    CL25' // LIBDEF PHASE,CATALOG='
LIBSLIB  DC    CL56'                '
         DC    CL81' // OPTION CATAL'
         DC    CL81'  INCLUDE'
         DC    X'FF'
RECAREA2 DC    CL81' /*'
         DC    CL81' // EXEC LNKEDT'
         DC    CL81' /*'
         DC    CL81' /&&'
         DC    X'FF'
PHASEX   DC    C',S+X''000000'''
PHASECD  DC    CL81'  PHASE '
ESDREC   DS    0CL81
         DC    X'4002'
         DC    C'ESD'
         DC    C'      '
         DC    X'0010'
         DC    C'  '
         DC    X'0001'
ESDRECN  DC    CL8' '
         DC    X'00'
ESDRECA  DC    X'000000'
         DC    C' '
ESDRECL  DC    AL3(0)
         DC    CL48' '
TXTREC   DS    0CL81
         DC    X'4002'
         DC    C'TXT '
TXTRECA  DC    AL3(0)
         DC    C'  '
TXTRECL  DC    XL2'0000'
         DC    C'  '
         DC    X'0001'
TXTRECD  DC    CL56' '
         DC    CL8' '
ENDREC   DS    0CL81
         DC    X'4002'
         DC    C'END '
ENDRECA  DC    AL3(0)
         DC    C'      '
         DC    X'0001'
         DC    CL64' '
PARMAREA DC    CL100' '
         LTORG
PUNCH    DTFDI DEVADDR=SYSPCH,IOAREA1=IO1,RECSIZE=80
CARDS    DTFCD DEVADDR=SYSIPT,TYPEFLE=INPUT,EOFADDR=LOPPU,IOAREA1=IO1, C
```

```
                WORKA=YES
IO1       DS    CL80
INCARD    DS    CL80
LEN       DS    F
FLEN      DS    F'0'
HLEN      DS    H'0'
REC5AL    DS    H'0'
MEMBNAME  DC    CL8'        '
TRTAUL    DS    0CL256
          DC    193X'00'
          DC    X'0A0B0C0D0E0F00000000000000000000'
          DC    32X'00'
          DC    X'000102030405060708090000000000000'
          DS    0F
MODAREA   DS    CL32000  COPY AS MANY AS NEEDED
MOD2AREA  DS    CL32000 COPY AS MANY AS NEEDED
          DS    CL32000
          END
/*
// EXEC LNKEDT,SIZE=256K
/*
/&
* $$ EOJ
```

C.10 Program to Create a Punch File from a Resource

This program punches a resource in a hexadecimal dump format in the POWER punch queue. The output file consists of 80-byte records. The hexadecimal representation is used to avoid the problems with the code conversion while transferring the punched output to the OS/2 system.

The punch file can be downloaded to the workstation using the Intelligent Workstation functions by entering the command RECEIVE pcfile (FILE=PUN.

C.10.1 IBM S/370 Assembler Coding for VSE

```
* $$ JOB JNM=RESTOOS2,CLASS=O,DISP=D,PRI=9
// JOB VSE2OS2P PUNCHING A RESOURCE INTO THE PUNCH QUEUE
// OPTION LINK,NOALIGN
// ASSGN SYSLST,FEE
* $$ PUN DISP=K,CLASS=A,JSEP=(0,N)
// ASSGN SYSPCH,PUNCH
// EXEC ASSEMBLY,SIZE=512K
RESPUNCH CSECT
         BALR 12,0                ESTABLISH
         USING *,12               ADDRESSABILITY
         LA   13,SAVEA            LOAD ADDRESS OF SAVE AREA
         TM   0(1),X'80'          IS THERE A PARAMETER RECORD?
         BNO  NOPARM              NO, THEN QUIT
         L    2,0(1)              LOAD ADDRESS OF PARAMETER AREA
         LA   1,2(,2)             LOAD ADDRESS OF PARAMETER DATA
         LH   2,0(2)              LOAD LENGTH OF PARAMETER DATA
         SH   2,=H'1'             DECREASE LENGTH BY 1
         EX   2,PARMMOVE          MOVE THE PARAMETER DATA
         CLI  PARMAREA,X'40'      IS THERE A NAME IN PARAMETER AREA
         BE   NOPARM              NO THEN QUIT
         MVC  FDEFNAME(8),PARMAREA  MOVE THE NAME
         LA   1,FDEFNAME          LOAD THE ADDRESS OF THE NAME
```

```
         CDLOAD (1),RETPNF=YES     ISSUE THE LOAD FOR THE PHASE
         LTR   15,15               WAS IT SUCCESSFUL?
         BNZ   NOPARM              NO, THEN QUIT
         ST    0,MODADDR           REQUESTED RESOURCE LOADED
         ST    1,MODENTRY          STORE REGISTERS
         ST    14,MODLENGT         ON THE TIME OF RETURN
         OPEN  PUNCH               OPEN THE PUNCH
         LA    6,IO1               LOAD IOAREA ADDRESS
         ST    6,CARDPOS           STORE CURRENT POSITION
         LA    1,80                80 BYTES PER CARD
         ST    1,CARDREM           STORE NUMBER OF BYTES LEFT
         L     3,MODADDR           LOAD PHASE ADDRESS IN GETVIS
         L     5,MODLENGT          LOAD PHASE LENGTH
TESTL    DS    0H
         LTR   5,5               LENGTH ALREADY = 0
         BZ    END               YES, THEN FINISH
         MVC   HLENGTH,0(3)      MOVE RECORD LENGTH
         LH    4,HLENGTH         LOAD INTO REG 4
         SH    4,=H'4'           SUBSTRACT THE LENGTH OF THE PREFIX
         LA    2,4(3)            LOAD THE ADDRESS OF THE 5A RECORD
         BAL   10,PUTPUNCH       PUNCH IT ONTO THE CARDS
         AH    3,HLENGTH         ADD LENGTH TO THE ADDRESS
         SH    5,HLENGTH         DECREASE REMAINING LENGTH
         B     TESTL             GO BACK TO TEST
END      DS    0H                END OF RESOURCE
         L     1,CARDREM         BYTES REMAINING ON THE CARD
         C     1,=F'80'          IS IT 80?
         BE    NOMORE            YES, DO NOT PUNCH EMPTY CARD
         PUT   PUNCH             PUT THE LAST CARD
NOMORE   DS    0H
         CLOSE PUNCH             SO THE JOB IS DONE
NOPARM   DS    0H
         EOJ
PUTPUNCH DS    0H                PUNCH THE RECORDS OF THE MODULE
         ST    5,PR5SAVE         STORE R5
         ST    4,PR4SAVE         STORE R4
         ST    2,PR2SAVE         STORE R2
         L     6,CARDPOS         RESTORE POSITION
         L     1,CARDREM         AND SPACE REMAINING ON CARD
PRLTR44  DS    0H
         LTR   4,4               LENGTH POSITIVE
         BNP   PRRET             NO, RETURN
         LA    5,0(2)            GET RECORD ADDR
         SR    7,7               CLEAR R7
NEXTCHAR DS    0H
         IC    7,0(5)            GET A CHARACTER
         SLL   7,24              CLEAR SECOND HEX DIGIT
         SRL   7,28              FROM THE CHARACTER
         STC   7,0(6)            STORE FIRST HEX DIGIT
         IC    7,0(5)            GET THE CHARACTER
         SLL   7,28              CLEAR THE FIRST DIGIT
         SRL   7,28              FROM THE CHARACTER
         STC   7,1(6)            STORE SECONF HEX DIGIT
         TR    0(2,6),=C'0123456789ABCDEF' CONVERT HEX TO CHAR
         LA    6,2(,6)           INCREASE POINTER ON THE CARD
         LA    5,1(,5)           AND IN THE INPUT AREA
         S     1,=F'2'           SPACE REMAINING DECREASED BY TWO
         BP    NEXTCHAC          CHECK THE NEXT CHAR
         PUT   PUNCH             CARD FULL, PUNCH THE CARD
```

```
         MVC   IO1(80),=CL80' ' CLEAR OUTPUT
         LA    6,IO1            RESET POINTER
         LA    1,80             AND SPACE REAMINING
NEXTCHAC DS    0H
         BCT   4,NEXTCHAR      CHECK IF STILL CHARACTERS IN INPUT
PRRET    DS    0H               RETURN
         ST    1,CARDREM        SAVE SPACE REMAINING
         ST    6,CARDPOS        AND POSITION on THE CARD
         L     5,PR5SAVE        LOAD THE
         L     4,PR4SAVE          SAVED
         L     2,PR2SAVE           REGISTERS
         BR    10               BRANCH BACK TO CALLING ROUTINE
PARMMOVE MVC   PARMAREA(1),0(1)
         DS    0F
IOC1     DS    0CL80
IO1      DC    CL80' '
SAVEA    DS    9D
FDEFNAME DS    D
CARDREM  DS    F
CARDPOS  DS    F
PR5SAVE  DS    F
PR4SAVE  DS    F
PR2SAVE  DS    F
MODADDR  DS    F
MODLENGT DS    F
MODENTRY DS    F
HLENGTH  DS    H
PARMAREA DS    0CL100
         DC    100C' '
         LTORG
PUNCH    DTFCD DEVADDR=SYSPCH,DEVICE=1442,IOAREA1=IOC1,BLKSIZE=80,     C
               TYPEFLE=OUTPUT,RECFORM=FIXUNB
         END
/*
// EXEC LNKEDT
// LIBDEF PHASE,SEARCH=PRD2.AFP
// EXEC ,PARM='T1000437'
/*
/&
* $$ EOJ
```

C.11 Program to Create the Resource from a Downloaded Punch File

This program creates the resource using the hexadecimal punch file created in the VSE system and then downloaded to the OS/2 system. The program receives the names of the input and output files as parameters. The output file is a resource that can be added to the library in PSF/2 by using the RLADD command.

C.11.1 C Coding for OS/2

```
#include <stdio.h>
#include <io.h>
#include <ctype.h>
#include <string.h>
FILE *stream;
FILE *stream2;
char namebuf[15];
```

```
char namebuf2[15];
char *name;
char *name2;
char testbuf[5]="0x   ";
main(argc,argv)
int argc;
char *argv[];
{
char *stopstring;
char buffer[10];
char ch;
int numread;
unsigned long il;
/* if input file name was not given obtain it */
if (argc > 1)
name = argv[1];
else {
printf ("Enter input file name:  ");
name = gets(namebuf);
}
/* if output file name was not given obtain it */
if (argc > 2)
name2 = argv[2];
else {
printf ("Enter output file name:  ");
name2 = gets(namebuf2);
}
if ((stream = fopen(name,"rb")) == NULL) return(1);
if ((stream2 = fopen(name2,"wb")) == NULL)  return(2);
/* get two hex digits (discard CRLF, end with blank) */
/* convert them to one character and write to the output */
hachar:
 ch=getc(stream);
if (ch==0x20) goto finish;
 testbuf[2]=ch;
 if (feof(stream2)) goto finish;
 ch=getc(stream);
 testbuf[3]=ch;
 if (testbuf[2]==0x0a) goto hachar;
 if (testbuf[2]==0x0d) goto hachar;
 if (feof(stream)) goto finish;
il=strtoul(testbuf,&stopstring,16);
buffer[0]=il;
numread=fwrite(buffer,1,1,stream2);
goto hachar;
finish:
return(0);
}
```

Appendix D. OS/400 AFP Sample Programs

This chapter documents two utility programs that we used for the OS/400 platform to print our test cases.

All of the coding documented in this chapter is presented as *sample coding only.*

Be sure that you have read the information in “Special Notices” on page ix.

The following table serves as an index to the various routines.

Name	Language	Description	Page
AS4002OS	C	This routine removes the extra blanks from an AFP resource transferred from the host system to an OS/2 system.	189
OS22AS4	C	This routine pads the AFP records in an AFP resource in the OS/2 system with trailing blanks.	190

D.1 AS4002OS Routine to Remove Extra Blanks

As AFP files are transferred from an AS/400 system, the files are padded with trailing blanks. PSF/2 does not accept resources in this format. This C language program removes the blanks from each AFP record, so the resulting resource is usablo in PSF/2.

D.1.1 AS4002OS C Program

```
#include <stdio.h>
#include <io.h>
#include <ctype.h>
#include <string.h>

      FILE    *stream;
      FILE    *stream2;
      char    namebuf[15];
      char    namebuf2[15];
      char    *name;
      char    *name2;
      char    ch;
      char    ch1;
      char    ch2;
      int     numread;
      int     i;
      int     j;
      char    buffer[80];
main(argc,argv)
int argc;
char *argv[];
{
       /* Get a file if one was not specified as an argument   */

       if (argc > 1)
        name = argv[1];
       else {
 printf ("Enter output file name: ");
```

© Copyright IBM Corp. 1994

```
        name = gets(namebuf);
       }
  if (argc > 2)
        name2 = argv[2];
       else {
        printf("Enter input file name: ");
        name2 = gets(namebuf2);
       }

       /* Open files in binary mode     */

       if ((stream = fopen(name,"wb")) == NULL)
        return (1);
       if ((stream2 = fopen(name2,"rb")) == NULL)\
        return (1);
 hachar:
   ch=getc(stream2);
    if (feof(stream2))
     {goto finish;}
    if (ch == 0x5a)
{
  buffer[0]=ch;
  numread=fwrite(buffer,1,1,stream);
  ch1=getc(stream2);
  ch2=getc(stream2);
  buffer[0]=ch1;
  buffer[1]=ch2;
  numread=fwrite(buffer,1,2,stream);
  j=ch1*256+ch2;
  for (i=1;i<j-1;i++)
   {buffer[0]=getc(stream2);
   numread = fwrite(buffer,1,1,stream);}
   }
   goto hachar;
finish:
      return (0);
}
```

D.2 Program to Pad a Resource with Blanks

Physical file members used for creating AFP resources in an AS/400 system have to padded with blanks. The file has also to have fixed length records. This routine pads records in an AFP resource in OS/2 with trailing blanks. The record size in this program is set to 16384 bytes, which should be enough to accommodate any AFP resource.

D.2.1 OS22AS4 C Program

```
#include <stdio.h>
#include <io.h>
#include <ctype.h>
#include <string.h>

     FILE    *stream;
     FILE    *stream2;
     char    namebuf[15];
     char    namebuf2[15];
     char    *name;
```

```
     char    *name2;
     char    ch;
     char    ch1;
     char    ch2;
     int     numread;
     int     i;
     int     j;
     char    buffer[80];
main(argc,argv)
int argc;
char *argv[];
{
       /* Get a file if one was not specified as an argument   */

       if (argc > 1)
        name = argv[1];
       else {
 printf ("Enter output file name: ");
        name = gets(namebuf);
       }
  if (argc > 2)
        name2 = argv[2];
       else {
        printf("Enter input file name: ");
        name2 = gets(namebuf2);
       }

       /* Open files in binary mode     */

       if ((stream = fopen(name,"wb")) == NULL)
        return (1);
       if ((stream2 = fopen(name2,"rb")) == NULL)\
        return (1);
 hachar:
    ch=getc(stream2);
    if (feof(stream2))
     {goto finish;}
    if (ch == 0x5a)
{
  buffer[0]=ch;
  numread=fwrite(buffer,1,1,stream);
  ch1=getc(stream2);
  ch2=getc(stream2);
  buffer[0]=ch1;
  buffer[1]=ch2;
  numread=fwrite(buffer,1,2,stream);
  j=ch1*256+ch2;
  for (i=1;i<j-1;i++)
    {buffer[0]=getc(stream2);
    numread = fwrite(buffer,1,1,stream);}
  for (i=1;i<16384-j;i++)
    {buffer[0]=0x40;
    numread = fwrite(buffer,1,1,stream);}
   }
    goto hachar;
finish:
      return (0);
}
```

List of Abbreviations

ACIF	AFP Conversion and Indexing Facility (PSF)
AFP	advanced function presentation (printing)
AFPDS	advanced function printing data stream
AIX	advanced interactive executive (IBM′s flavor of UNIX)
API	application program interface
APPC	advanced program-to-program communication
ASA	American Standards Association
ASCII	American National Standard Code for Information Interchange
BCOCA	Bar Code Object Content Architecture (IBM trademark)
CC	carriage control
CD-ROM	(optically read) compact disk - read only memory
CICS	customer information control system (IBM)
CMS	conversational monitor system (VM-based software, IBM)
CP	command processor
CRLF	carriage return/line feed
DOS	disk operating system (PC and 370 system)
FORMDEF	form definition
FTP	file transfer program
GDDM	graphical data display manager (IBM program product)
GOCA	graphics object content architecture
IBM	International Business Machines Corporation
ICCF	interactive computing and control facility
IEBCOPY	utility program (MVS)
IEBGENER	utility program (MVS)
IMS	information management system
IPDS	intelligent printer data stream (IBM)
IPS	include page segment (AFP command)
ISPF	interactive system productivity facility (MVS & VM)
ISPF	interactive structured programming facility
ITSO	International Technical Support Organization
JCL	job control language (MVS and VSE)
JECL	job entry control language
JES	job entry subsystem (MVS counterpart to VM′s RSCS)
KB	kilobyte, 1000 bytes (1024 bytes memory) case should be Kb
LAN	local area network
LPD	line printer daemon (AIX)
LPR	line printer control program and spooler (AIX)
LRECL	logical record length
MVS	multiple virtual storage (IBM System 370 & 390)
MVS/ESA	multiple virtual storage/enterprise systems architecture (IBM)
NJE	network job entry
OGL	overlay generation language
PAGEDEF	page definition
PC	Personal Computer (IBM)
PDM	printer driver machine
PM	presentation manager (SAA)
PNET	power networking interface
POWER	priority output writers, execution processor, and input readers (DOS)
PRPQ	programming request for price quotation (IBM custom built program products)
PSF	Print Services Facility (IBM program product)
PSF/MVS	Print Services Facility/MVS
PSF/VM	Print Services Facility/VM

© Copyright IBM Corp. 1994

PSF/VSE Print Services Facility/virtual storage extended

PTF program temporary fix

RDR reader

RECFM record format

REXX restructured extended executor language

RSCS remote spooling communications subsystem (VM's counterpart to MVS JES NJE)

SAA Systems Application Architecture

SCS SNA character string

SFCM spool file conversion machine

SMF system measurement facility

SNA systems network architecture (IBM)

TCP/IP Transmission Control Protocol/Internet Protocol

TRC table reference character (3800)

TSO time sharing option

UCS universal character set

VAFP virtual advanced function printer (VM)

VBA variable blocked (with ANSI carriage control characters)

VBM variable blocked (with machine carriage control characters)

VM virtual machine (IBM System 370 & 390)

VM/ESA virtual machine/enterprise systems architecture (IBM)

VSE virtual storage extended (IBM System/370)

VSE/POWER virtual storage extended/priority output writers, execution processors, and input readers (IBM)

XMIT transmit

Index

© Copyright IBM Corp. 1994

G

H

I

J

L

M

N

O

P

Q

R

S

T

V

W

X

ITSO Technical Bulletin Evaluation RED000

AFP Printing in an IBM Cross-System Environment

Publication No. GG24-3765-00

Your feedback is very important to help us maintain the quality of ITSO Bulletins. **Please fill out this questionnaire and return it using one of the following methods:**

- Mail it to the address on the back (postage paid in U.S. only)
- Give it to an IBM marketing representative for mailing
- Fax it to: Your International Access Code + 1 914 432 8246
- Send a note to REDBOOK@VNET.IBM.COM

Please rate on a scale of 1 to 5 the subjects below.
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction ____

Organization of the book	____	Grammar/punctuation/spelling	____
Accuracy of the information	____	Ease of reading and understanding	____
Relevance of the information	____	Ease of finding information	____
Completeness of the information	____	Level of technical detail	____
Value of illustrations	____	Print quality	____

Please answer the following questions:

a) If you are an employee of IBM or its subsidiaries:

Do you provide billable services for 20% or more of your time? Yes____ No____

Are you in a Services Organization? Yes____ No____

b) Are you working in the USA? Yes____ No____

c) Was the Bulletin published in time for your needs? Yes____ No____

d) Did this Bulletin meet your needs? Yes____ No____

If no, please explain:

_____

_____

What other topics would you like to see in this Bulletin?

_____

_____

What other Technical Bulletins would you like to see published?

_____

Comments/Suggestions: **(THANK YOU FOR YOUR FEEDBACK!)**

_____ _____
Name Address

_____ _____
Company or Organization

_____ _____
Phone No.

Cut or Fold
Along Line

Fold and Tape **Please do not staple** Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM International Technical Support Organization
Mail Station P099
522 SOUTH ROAD
POUGHKEEPSIE NY
USA 12601-5400

Fold and Tape **Please do not staple** Fold and Tape

Cut or Fold
Along Line

Printed in U.S.A.