Print and View Data Streams

IBM

International Technical Support Centers

Print and View Data Streams

GG24-3938-00

GG24-3938-00

Print and View Data Streams

Document Number GG24-3938-00

December 1993

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 xv.

First Edition (December 1993)

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
H52 Mail Station P099
522 South Road
Poughkeepsie, New York 12601-5400

Abstract

This document describes different data streams that are used in printing and viewing. It also deals with the transformation from one data stream to another.

This document was written for customers and IBM system engineers and system specialists who need information about these data streams.

No special prerequisite knowledge is required, but some general knowledge of data processing is assumed.

The contents of this document apply to the those releases of hardware and software that were available for the residency at that time.

There may be changes in the more recent releases of the software.

The reader is asked to find the current information in the product manuals and announcement letters.

ITSC Printing Library

This publication is part of the **ITSC Printing Library**.

LS MR PR PS VM (248 pages)

Contents

Figures

Tables

Special Notices

This publication is intended to help the customers and IBM system engineers and system sepcialists to understand the different data streams that are used for printing and viewing. The information in this publication is not intended as the specification of any programming interfaces that are provided by any of the program products mentioned in the document. See the PUBLICATIONS section of the IBM Programming Announcement for the referred products 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 information about non-IBM (VENDOR) products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. 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 Printing	AFP
AIX	Application System/400
AS/400	Bar Code Object Content Architecture
BCOCA	BookManager
BookMaster	CICS
Critique	DB2
DisplayWrite	Distributed Relational Database Architecture
DrawMaster	DRDA
Facsimile Support/400	GDDM
IBM	ImagePlus
Intelligent Printer Data Stream	IPDS
Micro Channel	MVS/ESA
OfficeVision	OfficeVision/MVS
OfficeVision/VM	OfficeVision/400
Operating System/2	OS/2
OS/400	Personal System/2
Presentation Manager	Print Services Facility
ProcessMaster	PROFS
Proprinter	PS/2
PSF	Quietwriter
SAA	SQL/DS
System/370	Systems Application Architecture
VM/XA	VTAM

The following terms, which are denoted by a double asterisk (**) in this publication, are trademarks of other companies:

Aldus	Aldus Corporation
Ami Pro	Samna Corporation
CADAM	Cadam Inc
CALCOMP	Sanders Associates, Inc.
CATIA	Dassault Systemes
Century Schoolbook	American Type Foundry
CompuServe	CompuServe Incorporated
Corel	Corel Systems Corporation
DEC	Digital Equipment Corporation
DXF	AutoDesk, Inc.
GEM	Digital Research, Inc.
Harvard Graphics	Software Publishing Corporation
Helvetica	Linotype Company
Hewlett-Packard	Hewlett-Packard Company
HP	Hewlett-Packard Company
HP PCL4	Hewlett-Packard Company
ITC Avant Garde Gothic	International Typeface Corporation
ITC Souvenir	International Typeface Corporation
LaserJet	Hewlett-Packard Company
Lexmark	Lexmark International, Inc.
Micrografx Designer	Micrografx Incorporated
Microsoft Windows	Microsoft Corporation
Monotype Garamond	The Monotype Corporation plc. (public limited company)
PageMaker	Aldus Corporation
Paintbrush	Z-Soft Corporation
Paradox	Borland International, Inc.
PostScript	Adobe Systems Incorporated
Professional Write	Software Publishing Corporation
Samma Word	Samma Corporation
Symphony	Lotus Development Corporation
Times New Roman	Monotype Corporation, Limited
UNIX	X/OPEN Company, Ltd.
Ventura Publisher	Ventura Software, Inc.
Windows	Microsoft Corporation

WordPerfect	WordPerfect Corporation
WordStar	MicroPro International Corporation
XEROX	XEROX Corporation
Xerox	Xerox Corp.
1-2-3	Lotus Development Corporation

Preface

This document describes data streams used for printing and viewing. A large part of the document deals also with the interrelationships between the different data streams. There are lots of programming examples to show how to convert the data streams to each other. Although some products are described in this document quite thoroughly, the document is not meant to be a product summary. The current information of the products mentioned can be found in the respective product manuals.

This document is intended for customers as well as IBM system engineers and system specialists.

How This Document is Organized

The document is organized as follows:

- Chapter 1, "Introduction"

  This chapter describes the purpose and structure of the document.

- Chapter 2, "Overview of Information Interchange Architecture"

  In this chapter there is an overview of the IBM Information Interchange Architecture (IIA).

- Chapter 3, "IBM Architectures"

  This chapter describes other IBM architectures related to documents as well as printing an viewing.

- Chapter 4, "Defined Standard Architectures"

  This chapter lists international standards for documents.

- Chapter 5, "IBM Data Streams"

  IBM data streams for different applications are described in this chapter.

- Chapter 6, "Industry Standard Data Streams"

  This chapter describes the industry standard data streams used in printing and viewing.

- Chapter 7, "Non-IBM Proprietary Data Streams"

  This chapter gives some examples of data streams that are neither IBM nor industry standard data streams, but are proprietary data streams used by some other companies.

- Chapter 8, "IBM Object Content Architectures and Definitions"

  This chapter describes in detail IBM Object Content Architecture.

- Chapter 9, "Defined Standard Object Content Architectures and Definitions"

  This chapter describes other standard object content architectures.

- Chapter 10, "Industry Standard Object Content Architectures and Definitions"

  In this chapter there are some of the industry standard object content architectures described.

- Chapter 11, "Document Languages and Formatting Languages"

This chapter deals with document and formatting languages.

- Chapter 12, "Print and View within the Different Environments"

  This chapter describes printing and viewing in different environments.

- Chapter 13, "Practical Tasks"

  This chapter includes a lot of practical examples a user may find when dealing with different data streams.

- Chapter 14, "Process Definitions"

  This chapter describes procedures to convert between different formats of data streams.

- Chapter 15, "Transforms"

  This chapter gives more examples of programs to be used for data stream conversions.

- Appendix A, "Source Code for Sample Transforms and EXECs"

  This appendix includes a lot of programming examples how to migrate from one data stream to the other.

- Appendix B, "Images, Graphics, and Data Streams"

  This appendix summarizes images and graphics data streams.

- Appendix C, "Products"

  This appendix has a short description of IBM products related to printing and viewing.

- Appendix D, "Products Involved in Printing and Viewing"

  This appendix has a table of the products is used in printing and viewing.

- Appendix E, "Bibliography"

  This appendix includes a list of related publications.

Related Publications

There is a list of publications in Appendix E, "Bibliography" on page 229. This list includes publications that contain more information about the topics covered in this document.

International Technical Support Organization Publications

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

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

The following chart shows the GBOF Code Form Numbers by which ITSO red books are shipped.

If you need help placing an order, or if you need assistance with adjusting your publications' subscription, your IBM Representative can assist you.

Category	GBOF Code
AIX Application Development and Database	6339
AIX Communications	6337
AIX Distributed Computing Environment	6342
AIX/ESA	6349
AIX Operating System/Systems Management & High Availability	6338
AIX Computer Graphics Series and User Interface (formerly called AIX 3D Computer Graphics Series)	5216
Application Development Platform	6321
Application Development Design and Modeling	6323
Application Development Maintenance and Test	6322
Application Generators	6324
Architecture	6360
Artificial Intelligence and Knowledge Based Systems	6326
AS/400 Communications and Systems Management	5225
AS/400 Office and Advanced Technology	5224
AS/400 PC Support	5223
AS/400 Systems, Application Development and Performance	5222
Automated Operations	6351
Banking - ATM	6352
Banking - Consumer Transact	6354
Banking - LAN DP	6353
Bibliography (SLSS by Form Number Only)	GG24-3070
CICS	6328
Communication Controller Products	5207
Connectivity	5208
Data Delivery and Information Warehouse	6331
DB2	6330
Distributed Applications	6333
Distributed Data Base	6332
Engineering and Scientific	2200

Category	GBOF Code
Enterprise Networking	5210
High Level Languages	6325
Host Systems Management	5204
IBM OS/2 Ext Ed Cookbooks	2195
IBM OS/2 V2.0 Remote Installation Maintenance	2224
IBM OS/2 Version 2.0 / 2.1 Technical Compendium	2254
IDNX	6362
ImagePlus/MVS	6318
ImagePlus/400	6319
ImagePlus/2	6320
IMS	6329
Large Systems Hardware	5200
Local Area Networks	6367
Miscellaneous/Cross Category	0429
MVS/ESA Open and Client/Server	6334
MVS/ESA Systems Products (formerly Large Systems Software)	5201
NCP	6364
NetWare for AIX	6341
NetWare for OS/2	6336
Network Distribution	6356
Network Management	5209
Network Performance	6355
Network Security	6358
Object Oriented Technology	6327
Open Networking	6359
Office Systems - AIX Systems	6346
Office Systems - Client/Server Systems	6348
Office Systems - Cross Systems	6347
Office Systems - LAN	6343
Office Systems - MVS Systems	6345
Office Systems - VM Systems	6344
OSI	6366
OS/2 Communications	6370
OS/2 LAN and Distributed Systems	6335
OS/2 V1.X	6301
OS/2 V2.X	6303
Personal Systems - Configuration, Installation, Distribution (CID)	6304
Personal Systems Hardware	6300
Personal Systems - Multimedia	5212
Personal Systems Software & Application Development	6302
Printing	5202

Category	GBOF Code
Retail	5214
RISC/6000 Hardware	6340
Storage Hardware - Magnetic DASD	6306
Storage Hardware - Optical DASD	6308
Storage Hardware - Tape	6307
Storage Software - DFSMS/MVS	6309
Storage Software - DFSMS/VM	6310
Storage Software - Distributed Storage Management	6311
System/Network Design	6371
Systems Application	6357
Systems Solution Library	6372
System Management Reference Library	6373
TCP/IP	6368
VM Systems	2201
VM/VSE	6313
VM/VSE - CSP	6317
VM/VSE - SQL	6314
Voice Enablers	5211
VSE	6312
VTAM	6363
X.25	6369
3174	6365
937X	6361

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:

Mike Calder-Smith
IBM United Kingdom

Rudolf Hochscheid
IBM Germany

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

The authors would also like to thank:

Paul Jackson
IBM Hursley Laboratory, UK

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 P402
522 SOUTH ROAD
POUGHKEEPSIE NY
USA 12601-5400

Fold and Tape **Please do not staple** Fold and Tape

ITSO Technical Bulletin Evaluation RED000

Print and View Data Streams

Publication No. GG24-3938-00

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.

Chapter 1. Introduction

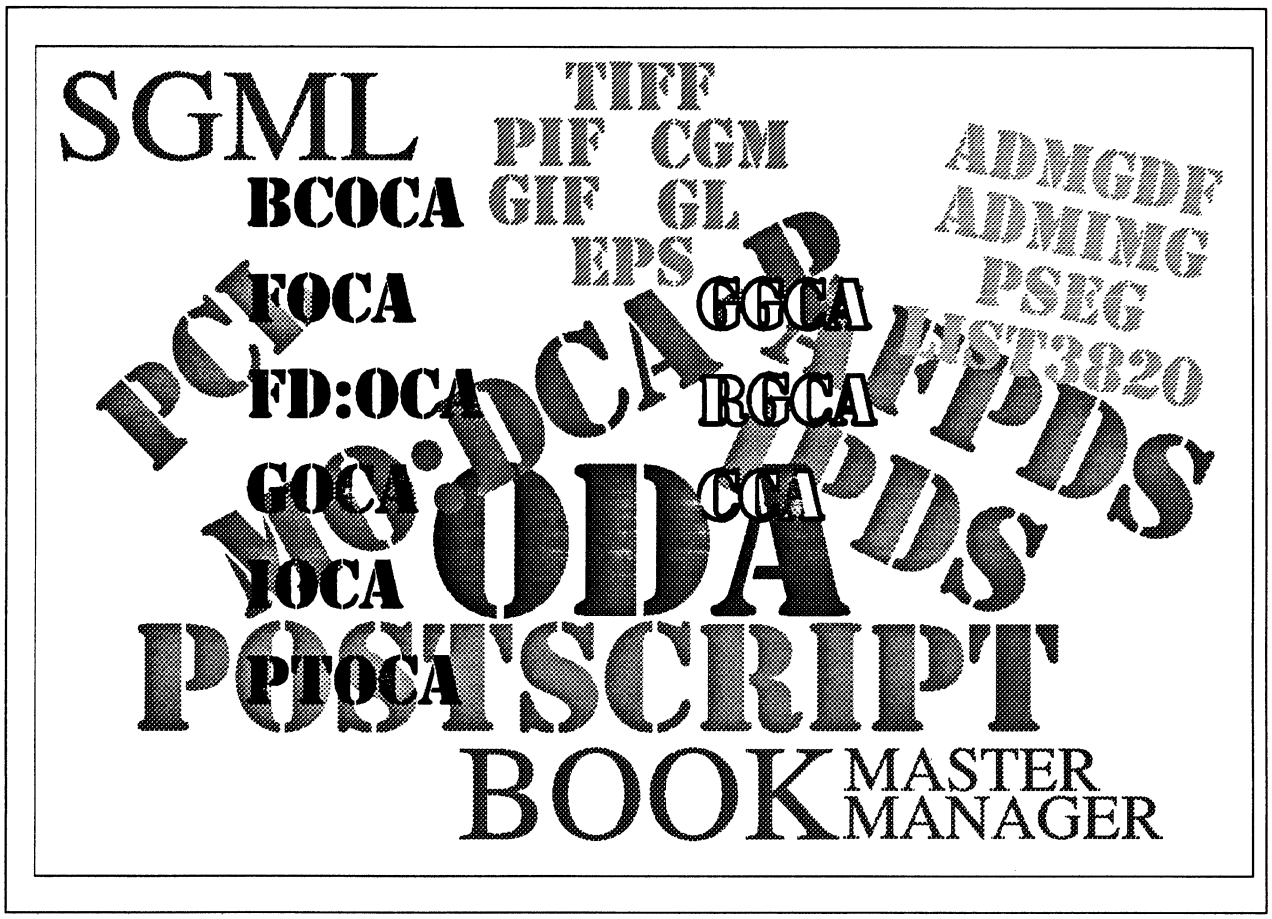


Figure 1. Just a Few of Our, Well, Not Exactly Problems ...

1.1 Objectives

The primary objective of this document is to clarify the interrelationships between different print and view data streams to help make it easier to understand which products will work together properly.

This can be shown by:

- Outlining what type of data is contained in each data stream.
- Outlining how they relate to each other and the I/O devices that will be used to display or print the data.
- Indicating which of the most commonly used programs produce and use which types of data streams.

Clearly, these objectives are very wide ranging, and we must restrict ourselves to the most commonly used data streams. The book will consider only those data streams in common use in publishing and printing, and which have significant use on IBM* platforms.

This will, for example, exclude such formats as IGES, which is more specific to the CAD (Computer Aided Design) arena, and such as proprietary typesetting data streams will be effectively ignored as being specific to particular manufacturer's output devices.

The book is divided logically into two parts:

- The first chapters introduces the major IBM architectures, data streams, and objects, some international standards, and some data streams and objects which are commonly used in the data processing industry in general.
- The last chapters show how they can be used, relationships between them, and common programs that use them.

  This volume takes the approach that these aspects are best introduced and explained by use of practical examples, so we use specific user examples to explain how particular products are used to change and use data streams and objects.

You should not expect this volume to give a complete and detailed definition or description of data streams, architectures, data objects, programs, or means of transforming from one form to another; to do so would be to duplicate existing material. Where possible, we refer to documents which give that detailed information. The purpose of this document is to introduce the relationships, and indicate known paths though this complex area.

Finally, there are many situations where the user has data in one format or data stream, and needs it in another form, or to be output on a device that needs some other data stream or format, for which there is no existing process for conversion.

In this case, if the user need is to be met, a special conversion program, or *transform* must be written. Chapter 15, "Transforms" on page 135 introduces some of the principles and considerations to be taken into account when designing transforms, and outlines the factors which will affect the degree of success that can be expected from the transform.

1.2 Terms with Special Meanings

So that we do not have to continually qualify our descriptions, and to make the text more concise, we use some terms in specific ways in this book, which may not be the meanings you normally associate with those phrases.

The glossary should contain most technical words and phrases used here but not in common use, but these definitions are largely those accepted by the technical community.

Before reading this book, you should note the specific meanings we intend for the phrases below.

Data stream This is a phrase generally used to mean several things. In this book, one particular meaning is intended: a definition of format or formats that can be used to specify the content of a data file or a data stream along a communications link, which is capable of being used to contain a composite document.

(A composite document is one which may contain text, font, vector graphic, and image data — and possibly other things as well.)

In this book, we use the phrase *data stream* to refer both to these composite formats and also to instances of them.

Data Stream formats may be defined in *Data Architectures*.

Data object We needed a phrase to describe an instance of a data file or stream in a single format (text only, vector graphic only, and so on) which could either stand on its own or be a component of a data stream. In this book we use the phrase *data object* in this restricted sense.

Data object formats may be defined in *Content Architectures*.

Composite document A composite document is a document which may contain multiple types of data object, for example, vector graphic artwork as well as alphabetic text. A document in this sense need not be expressed on paper; the phrase can refer to a data set which could be used to create a paper document, or indeed the expression of it in softcopy form.

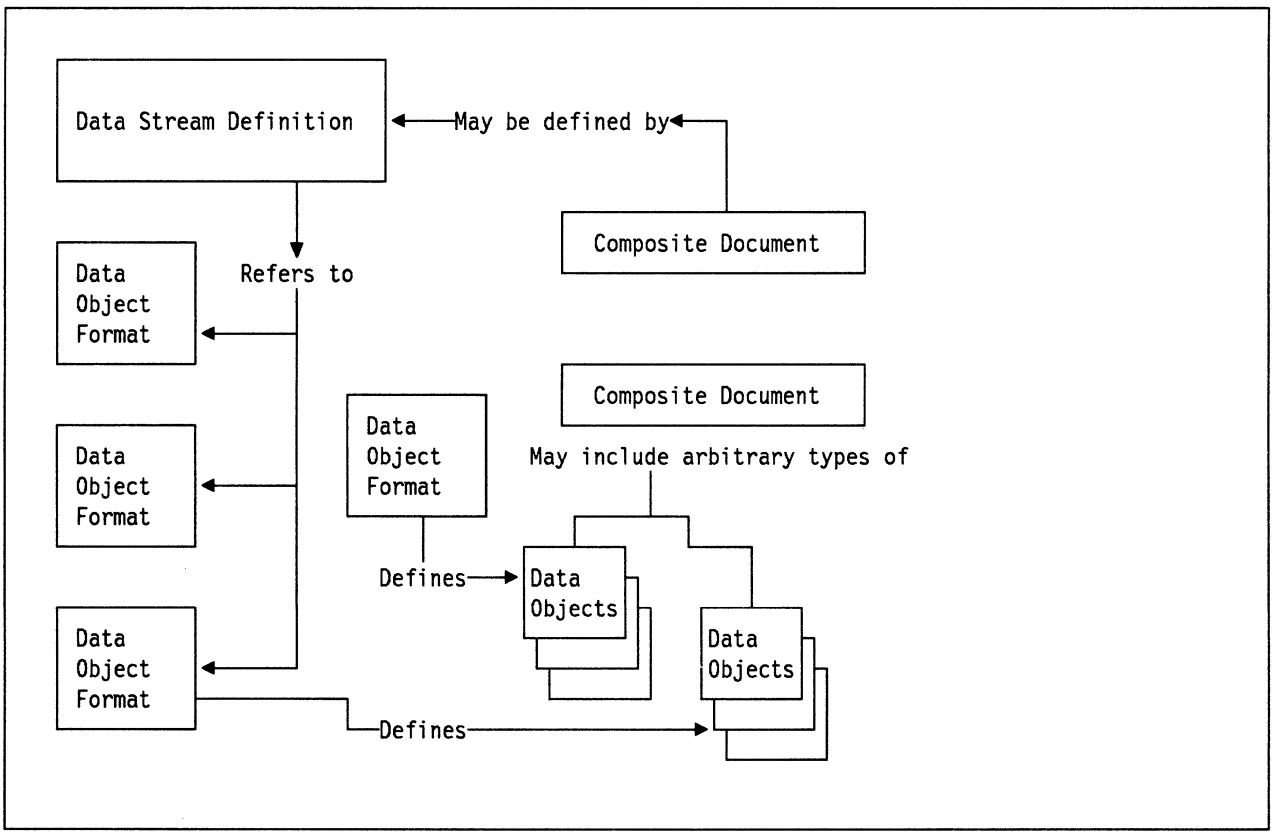


Figure 2. Relationship Between Data Streams, Objects, and Composite Documents

Interchange	The process of providing a document to a receiving person or device, by means of data communication or by exchange of storage media.
Editing	The carrying out of operations associated with creation and amendment of the structure and/or the content of a document.
Formatting	The carrying out of operations to determine the layout of a document, that is, the appearance of its content on a presentation medium.
Presentation	The operation of rendering the content of a document in a form perceptible to a human being. Typical presentation media are paper and video screens.

1.3 Overall Structure

There are many terms used to describe this area of data streams, and IBM terminology is often different from other parts of the industry. Competitive data streams can also have different types of relationships. In this book we attempt to divide the subject into the following:

IBM offerings

Defined standards

Industry standards

Non-IBM proprietary offerings

Under each of these, we consider architectures, data streams, and data objects.

The distinction we make between Defined Standards and Industry Standards is that Defined Standards are the subject of a published standard prepared by some national or international body such as ISO or ANSI, while Industry Standards are generally accepted within the industry as de facto standards and are used by many suppliers. Needless to say, there is often argument about the latter.

These cannot be hard and fast distinctions; one man's Industry Standard is another man's competition; PostScript can be considered a data stream or a data object, and so on.

However, we need some kind of a framework, and so long as we remember that some boundaries can be fluid, this can be a useful classification.

At the architecture level, we will place most emphasis on IBM and Defined Standard architectures, as being of most interest to our audience. Competitive and Industry Standard areas are dealt with more at the data stream and data object level; one tends to attempt to convert or use a document or data file from one system or format to another in practice. Considerations of relative architectural structures or merits are more of academic or marketing interest.

For people who want an overview of particular architectures, data streams, and file formats, the first part of this book contains such material. The information in these chapters is also cross-referenced to where appropriate by the task-related sections which follow.

1.4 Print and View: the Messages, the Opportunities

Point 1 *Every application is an AFPDS application.*

With PSF/2 and the IBMAFP printer drivers we can integrate printing from host and workstation applications.

It is well understood how to migrate host applications to AFP*, but most people don't realize that now every OS/2 and DOS/Windows application can create an AFPDS just by setting up a simple printer driver.

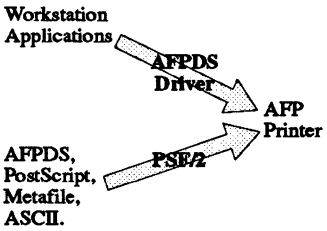
This means that every workstation application can now take advantage of the speed, volume, and integrity benefits of AFP, while maintaining the quality of output they are used to.

EVERY APPLICATION

is an

AFPDS

APPLICATION.



Point 2 *Every printer is an AFPDS printer.*

The AFP Workbench for Windows application isn't just a way of looking at AFPDS files before you print them. As a DOS/Windows application, it can take advantage of *every* printer driver that there is for DOS/Windows That means that *any* workstation printer can now print AFPDS data streams! All you have to do is load it into AFP Workbench for Windows, and print it off, using the appropriate driver.

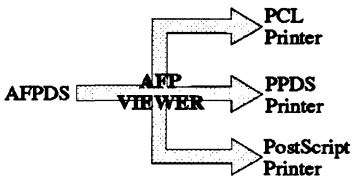
Not only that, but the drivers can be "connected" to a file, so AFP Workbench for Windows can be used to convert AFPDS to PostScript**, PPDS, PCL, you name it.

EVERY PRINTER

is an

AFPDS

PRINTER.

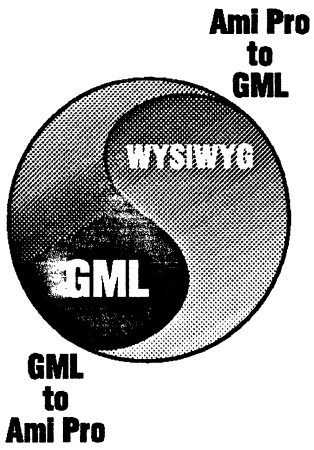


Point 3 *Ami Pro** to GML to Ami Pro to GML to ...*

Using the ASCII import/export with style names functions of Ami Pro, and a couple of simple workstation programs, we can go round this cycle as much as we like, preserving the vast majority of our structure and formatting information. There are limitations; we can't preserve all table structures, for example, but we can use Ami Pro as a WYSIWYG front end to GML, and use Ami Pro as a workstation GML formatter for a very wide range of GML documents.

It also means we can:

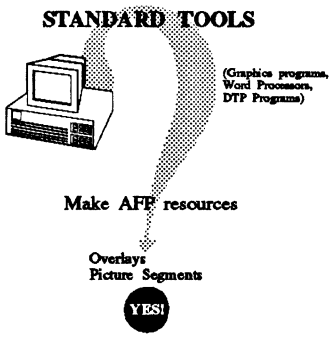
- Use a WYSIWYG workstation editor for GML documents.
- Format and AFP print GML on the workstation.
- Combine workstation documents into mainframe publishing.
- Archive workstation documents on the host using BookManager* READ, and use READ's powerful free text search on the archive.



Point 4 *AFP Resources from standard tools*

Until recently, it has been either awkward to make such things as overlays, or you have had to get expensive, specialized software. With the IBMAFP drivers, or AFP Workbench for Windows, you can now take a graphic or page layout from virtually any workstation graphic program, or word processing program, or DTP program, and create an overlay immediately. A PSEG is just as easy.

All you have to do to use them on the mainframe is upload, and run a simple record reblocking EXEC against them, to split the file into records.



Chapter 2. Overview of Information Interchange Architecture

Information Interchange Architecture (IIA) is IBM's way of defining a structure for information of all kinds, to clarify where particular formats of data relate one to another. It encompasses both IBM-specific elements and some international standards, so we give a brief overview here before looking at the parts of it in more detail.

2.1 Introduction to IIA

Below are some general concepts of IIA:

- IIA consists of architectural components.
- These components are used for the interchange of composite documents.
- Composite documents can contain mixtures of text, images, and graphics.

The architectural components consist of:

- Standard Generalized Markup Language
  SGML
- Office Document Architecture
  ODA
- Revisable Form Text/Document Content Architecture
  RFT:DCA
- Mixed Object: Document Content Architecture
  MO:DCA

These architectural components can define document-carrying data streams for interchange, and content architectures that define the data objects that make up documents

- Within IIA, data and attributes are separated from the document description. This minimizes data stream transformations within a system and makes it possible to support complex combinations of different data types, resources, and document descriptions in application-specific data streams as well as in device-specific data streams.
- The structure of IIA is open ended thus providing for functional growth in the future.
- Applications making full use of IIA and its architectures are more effective in communication, are working together, and are more efficient in a system and across a network.

More information about Information Interchange Architecture components under IIA can be found in the ITSO red book: *Information Interchange Architecture: Concepts, GG24-3503.*

2.1.1 Relationship of Architectures and Data Streams under SAA

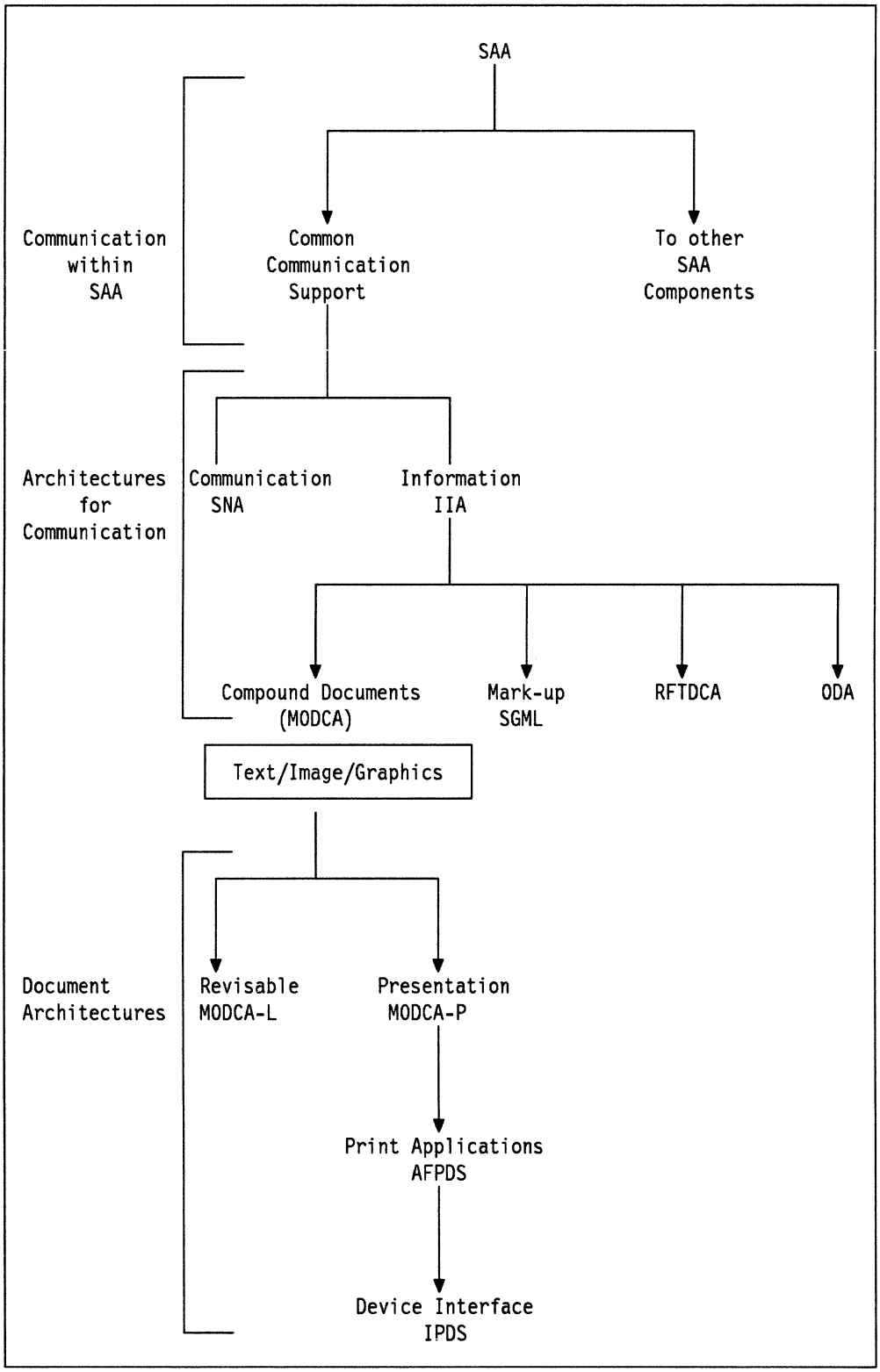


Figure 3. Overview of Architectures under IIA. This figure shows the relationships between document architectures and content architectures.

2.1.2 IIA Components

Normally users don't care what languages or architectures they use for their work; they know or learn how to handle available tools to produce a printed document on a printer they have access to.

The world of printing and communication has changed. Documents need to be printed on any printer in an enterprise. Wherever printing occurs, the end user expects to get the same page layout, the same data, and the same positioning of the data regardless of where the printer is physically located.

This means the users, or more likely whoever supports them, has to deal with a new world. The rest of this chapter introduces where things fit in an enterprise wide solution for printing and viewing.

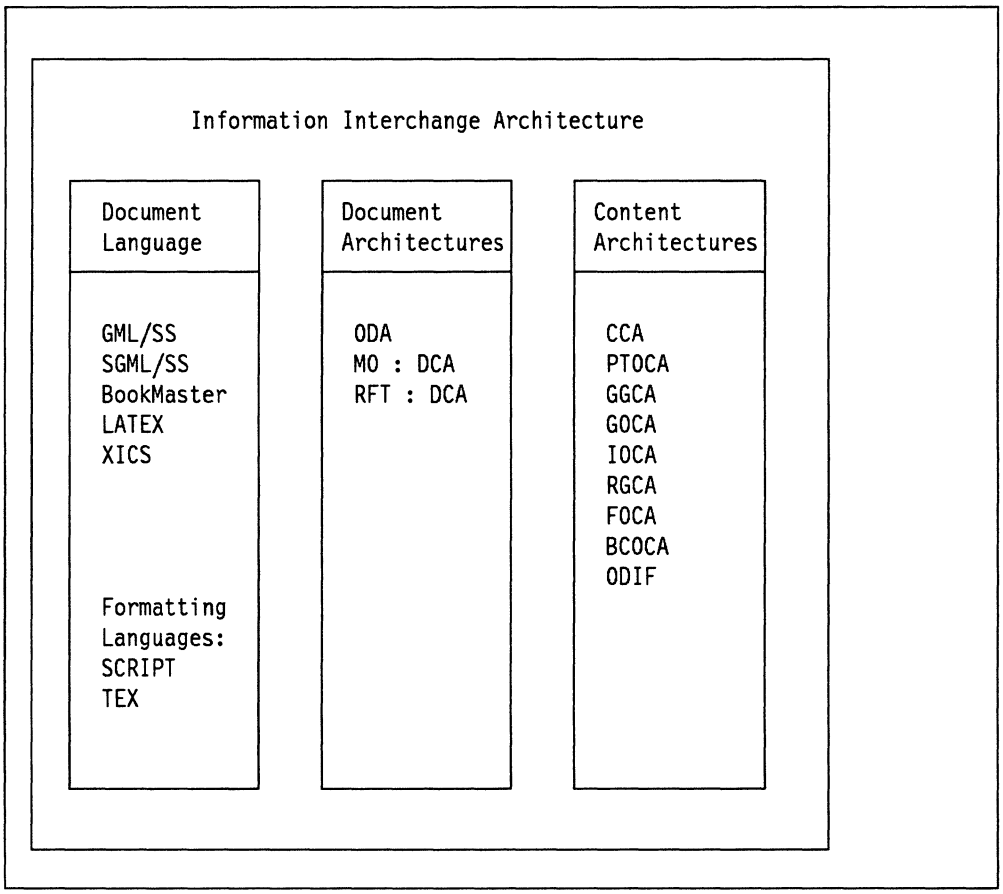


Figure 4. Overview of Information Interchange Architecture. This figure shows the main components of IIA, and what document architectures and content architectures are part of IIA.

2.1.3 Document Languages

Document Languages are device-independent and application-independent ways of defining the source content of composite documents. They are large, as they are intended to be humanly readable and editable. Before documents written in a document language are printed, they must be processed by some kind of computer program into a data stream that the printer to be used can interpret to produce the finished document.

A brief description is given here; a fuller one in Chapter 11, "Document Languages and Formatting Languages" on page 65.

2.1.3.1 GML (Generalized Markup Language)

Within IBM production publishing are two major document language families: GML and SGML. Their data formats are tag-based. Adding tags to text and data defines structural elements of the document, which are then formatted by a computer program according to rules defined in a *style* or profile.

The source document thus doesn't contain formatting information, as opposed to most documents created by office systems or word processors, where the user often decides on the formatting while typing the text into the system.

This means that documents written in a document language are device independent, and can be produced in a variety of formats just by changing the style or profile. Equally, use of standard house styles ensures that formal documents are produced in a consistent high quality format.

IBM Generalized Markup Language is a part of Document Composition Facility (DCF). IBM BookMaster* is an extended markup language which uses the DCF formatter to give a richer set of document elements and document styles.

DCF is a licensed program that is used in the preparation of printed documents. It can also process documents that are written in the IBM Script formatting language; in fact currently GML and IBM BookMaster are implemented using macros written in the IBM Script language, and text programmers can produce enterprise specific tags by writing new macros and adding them to the system libraries.

2.1.3.2 SGML (Standard Generalized Markup Language)

SGML is an international standard that builds on the original definition of markup languages. It provides a standard for the definition of markup languages, and provides for document structure control as well as the definition of document elements.

It is intended to provide a standard for information interchange between different computer processing systems and platforms.

The Standard Generalized Markup Language (SGML) Translator allows DCF to be used to format SGML data, by either of the following:

- Translating CALS SGML into a DCF understandable form
- Providing an SGML "Starter Set" of tags and macros to format them.

As with GML or IBM BookMaster, the use of SGML in an enterprise can be extended by a text programmer providing macros to format particular SGML tags.

2.1.4 Document Architectures

Document Architectures are ways of specifying the definition of data streams that can contain composite documents. Usually they refer to formatted data of some kind, unlike Document Languages.

The intention behind defining Document Architectures is to provide a consistent basis for the production of composite documents, and to provide an application

and device independent means of interchanging data between different users, applications, and computer systems.

They cannot provide total device independence, unlike Document Languages, because format definition always implies some specific capability on the part of the output device; however Document Architectures are usually aimed at either some *family* of output devices, or subsets of function that are expected to be available on the *type* of output device selected.

2.1.4.1 ODA

Office Document Architecture (ODA) is an ISO data stream architecture for interchanging revisable and presentation documents. The ODA standard provides controls for describing both the logical view and the layout view of a document.

An ODA document has the following perspectives:

- The content of the document, which is again divided into content portions
- The logical structure, which is a hierarchy of logical objects
- The layout structure, which is a hierarchy of logical layout objects

Logical objects are subdivisions of the document based on meaning.
Layout objects are subdivisions of the document based on appearance.
Content portions are associated with both kinds of objects.

ODA provides three classes of document architecture that represent the revisable form of a document as well as the presentation form:

1. Formatted
2. Processable
3. Formatted and processable (not included in IIA)

Additionally ODA provides architectures at the document level. Currently it provides architectures for text, image, and graphics, and it is extendable for other types.
However, ODA is useful for a standardized interchange on both a document level and at content level.

2.1.4.2 RFT:DCA

Revisable Form Text/Document Content Architecture (RFT:DCA) is an IBM data stream architecture for interchanging revisable documents. A RFT:DCA document is a combination of content and format information. A recipient of a RFT:DCA document can modify both the content and the format.

Historically, RFT:DCA has been a text-oriented architecture, but extensions have been made available for including data of non-text types.

Format information includes the general characteristics of a document contained in format declarations such as:

- Page width
- Page depth
- Page numbering scheme

These format declarations are separate from the text of the body of the document. They contain the information required to modify the overall structure of a document without disturbing the text.

Office Systems may be dissimilar and they may therefore offer different capabilities. RFT:DCA supports interchange among dissimilar office systems in three environments:

The Text Revision environment — RFT:DCA allows the text in a document to be modified at the receiving location.

The Format Revision environment — RFT:DCA allows the format information in a document to be modified to fit conditions at the receiving location.

The Final Format environment — FFT/DCA allows documents to be presented only as intended by the originator.

2.1.4.3 MO:DCA

Mixed Object: Document Content Architecture (MO:DCA) defines a mixture of different types of information within a single document. This is as opposed to FFT/DCA and RFT:DCA that define the interchange of text only documents.

MO:DCA contains text, image, graphics, and layout structures within one document.

MO:DCA divides into several interchange subsets:

1. MO:DCA-P is a final form data format to be printed or displayed.
   P stands for Presentation.
2. MO:DCA-L contains resource documents (mostly images and graphics) to be stored for a later reference by presentation data streams.
   L stands for Library. It normally shows up within ImagePlus*.
3. MO:DCA-R contains resource documents to be stored for a later reference by presentation data streams.
   R stands for Resources. An overlay is often treated as a MO:DCA-R document, since it is referenced to be included into an AFPDS data stream.

2.1.5 Content Architectures

In order to describe how a Document Content Architecture builds up, here are a few definitions of some terms used when describing IBM MO:DCA:

Document — A document is a machine-readable collection of one or more objects which forms a complete composition.

Document Element — A Document Element (also known as a *Structured Field*) is a record of variable length which is *self identifying*. It contains either control information or data or both.

Structured Field — Another name for a Document Element.

Document Component — A Document Component is a set of related structured fields which are bounded by structured fields marking the beginning and end.

Document Content Architecture Within a Document Content Architecture there is a family of *Content Architectures* that define the syntax and semantics of the different document components which are defined to be part of the *document content architecture data stream*.

2.1.5.1 Who Carries What

Document Content Architectures define what kind of data objects can be carried by the data stream.

MO:DCA-P can carry:

- FOCA
- GOCA
- IOCA
- PTOCA
- BCOCA*

ODA can carry:

- GGCA
- RGCA
- CCA

For a description of the MO:DCA content architectures refer to Chapter 8, "IBM Object Content Architectures and Definitions" on page 49.

For a description of the ODA architectures refer to Chapter 9, "Defined Standard Object Content Architectures and Definitions" on page 59.

Chapter 3. IBM Architectures

Table 1. Architectures, Content Architectures, and Data Streams under SAA. This table shows which print and view architectures and data streams are applicable under which operating system.

Architectures	*OS/2*	*AS/400*	*VM*	*MVS*	*IMS*	*CICS*
RFT:DCA	X	X	X	X	X	X
MO:DCA-P		X	X	X	X	X
Content Architectures						
PTOCA		X	X	X	X	X
IOCA	X	X	X	X	X	X
GOCA	X	X	X	X	X	X
BCOCA		X	X	X		
FOCA	X	X	X	X	X	X
FDOCA		X	X	X	X	X
CDRA			X	X	X	X
Data streams						
AFPDS	X	X	X	X	X	X
IPDS	X	X	X	X	X	X
3270DS		X	X	X	X	X
SCS		X	X	X	X	X
ASCII	X					

Note: Architectures are introduced in chapter Chapter 2, "Overview of Information Interchange Architecture" on page 9.

Content architectures are covered in more detail in chapter Chapter 8, "IBM Object Content Architectures and Definitions" on page 49.

Data streams are covered in more detail in chapter Chapter 5, "IBM Data Streams" on page 29.

3.1 Mixed Object Document Content Architecture

The Mixed Object Document Content Architecture (MO:DCA) is designed to integrate the different OCA data objects into a single, device-independent data stream that can be interchanged between application programs. The programs can be in the same system or on different systems. MO:DCA defines a memory and storage format for the data that is independent of communications protocols. Thus, the MO:DCA data stream can be interchanged across communications lines or through a common storage medium such as tape or disk.

The MO:DCA data stream consists of the following components:

Layout structure defines the way objects should be presented.

Objects (such as text, graphics, image, and bar code) define the pieces of a document.

Mapping specifies the relationship between the layout structure and objects.

Because a document's layout structure and objects are separate in the MO:DCA data stream, a change in one does not affect any other.

All functions and data that make up a MO:DCA data stream are contained in logical records called *structured fields*. Related structured fields are grouped into categories and bound by unique "begin" and "end" structured-field delimiters.

3.1.1 Minimum Functions Required for MO:DCA

The minimum functions required for interchangeable MO:DCA data streams are based on the intended use of the data stream and are defined by *interchange sets (IS)*. Currently, three interchange sets are defined—two for presentation (MO:DCA-P IS/1 and MO:DCA-P IS/2) and one for library (MO:DCA-L).

Within these interchange sets, the minimum functions required are dependent on the product class—generator, receiver, or receiver/generator. A *generator* is a product that creates data streams, while a *receiver* is a product that interprets and processes existing data streams.

In general, a generator is only required to generate a valid subset of an interchange set. A receiver must be capable of interpreting and processing all of the MO:DCA constructs contained in the interchange set and all of the constructs for at least one of the data objects contained in the interchange set. While a product classified as a generator may be able to receive its own interchange set subset, only those products that fully meet the requirements for both a generator and a receiver are classified as receiver/generator products.

A MO:DCA-P data stream is intended for presentation at a workstation or on a printer. MO:DCA-P IS/1 implementations must support all of the base MO:DCA constructs contained in the interchange set and at least one of the following objects:

- Presentation Text (PT1)
- Graphics (DR/2V0)
- Image (FS10)

MO:DCA-P IS/2 implementations must support all of the base MO:DCA constructs contained in the interchange set and at least one of the following objects:

- Presentation Text (PT1)
- Graphics (DR/2V0)
- Image (FS10 or FS11)
- Bar Code (BCD1)

A MO:DCA-L data stream is intended to save data (usually in a library) for later use by an application program. For example, the Presentation Manager* creates a MO:DCA-L data stream known as a *metafile* by means of calls to its SAA* presentation interface. MO:DCA-L implementations must support all of the base MO:DCA constructs contained in the interchange set, as well as the following objects:

- Graphics (DR/3V1)

- Image (FS20)

More detailed information about MO:DCA can be found in *Mixed Object Document Content Architecture Reference, SC31-6802.*

3.2 Revisable-Form-Text Document Content Architecture

Revisable-Form-Text Document Content Architecture (RFT:DCA) defines the structure of a text document that is in a form that can be edited or later formatted. Each recipient of a revisable-form document can modify its contents and format.

An RFT:DCA data stream consists of format units, text units, and an end unit.

- *Format units* contain format declarations and include no text except top and bottom margin text.
- One or more *text units* contain the body of the document.
- The *end unit* identifies the end of the document.

3.2.1 Minimum Functions Required for RFT:DCA

The minimum functions required for RFT:DCA interchange are:

- Format unit 1
- Format unit 2
- One or more text units
- An end unit

A text unit must contain at least one body-text structured field.

Additional information about RFT:DCA is in *Document Content Architecture: Revisable-Form-Text Reference, GG23-0758.*

Chapter 4. Defined Standard Architectures

Architectures in this category are defined by national and international standards bodies. In this case, the responsible body is International Standards Organization, and the standards that will be addressed, albeit briefly, in this section are:

1. Standard Generalized Markup Language (SGML) - ISO 8879
2. Document Style Semantics and Specification Language (DSSSL) - ISO Draft Standard DIS 10179
3. Standard Page Description Language (SPDL) - ISO Draft Standard DIS 10180
4. Office Document Architecture (ODA) - ISO 8613

SGML is described in overview in Chapter 11, "Document Languages and Formatting Languages" on page 65, particularly in 11.6, "SGML" on page 72.

DSSSL and SPDL are as yet just draft standards, and there are few if any applications which are based on them at the time of writing. Given the thrust towards open systems, and the success to date of SGML, it is unlikely that anyone in the industry can afford to ignore them. The only question is the time scale over which they will become significant.

Because of this, a very slight overview of them and their relationship with SGML is given below.

ODA has already been briefly referred to in Chapter 2, "Overview of Information Interchange Architecture" on page 9 as it is a component of Information Interchange Architecture. It is described in a little more detail as an international standard below.

4.1 The relationships between SGML, DSSSL, and SPDL

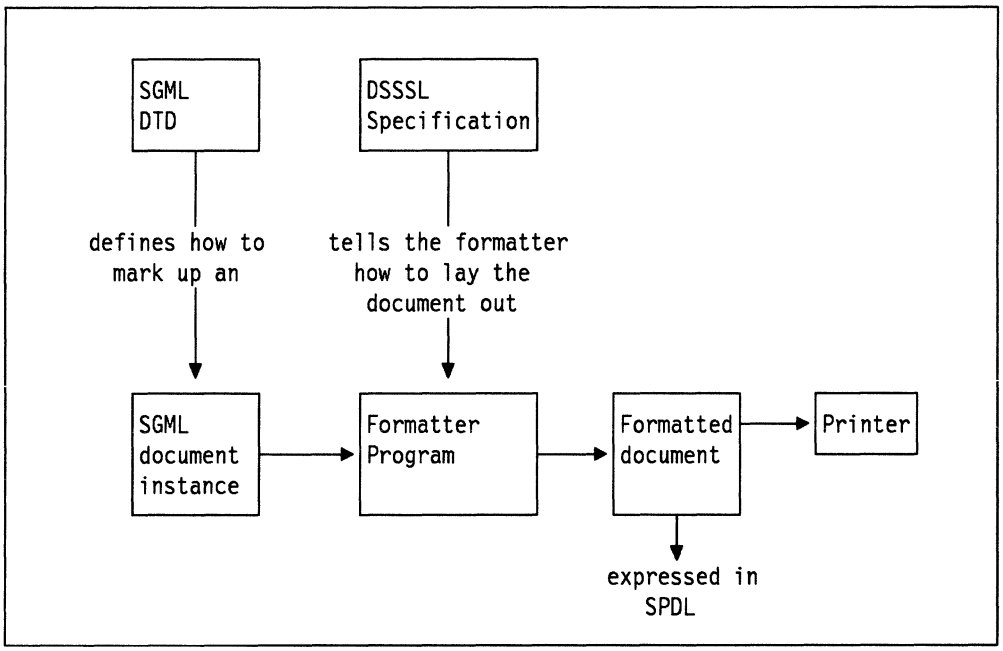


Figure 5. SGML, DSSL, and SPDL

SGML is the ISO document language. A document expressed in a document language must be formatted by a formatting program. For the formatting program to do its job, there must be expressed somewhere a set of rules which define how the document elements described in the document language are to be formatted. In IBM BookMaster, for example, these rules are contained in a macro library written in the IBM Script formatting language.

DSSSL is the ISO standard which defines how these rules are to be expressed, so that SGML-complying formatters can convert the source text to a final form.

The final form produced by formatters is often a data stream that is only understood by particular output devices. Many IBM programs use AFPDS as a final form; many workstation programs use PostScript.

ISO are in the process of defining a "Page Description Language" which is intended to become a standard final form for interchange. It may be used either directly by output devices, or possibly it may be converted to another print or view data stream for final presentation.

The ISO PDL is to be SPDL. It is based on the concepts which have proved so successful in the PostScript Page Description Language, and can in many ways be considered a superset of that language.

4.2 Office Document Architecture

Office Document Architecture (ODA) is an ISO data stream architecture for interchanging revisable and presentation documents. **The ODA standard provides controls for describing both the logical view and the layout view of a document**. The ODA document model has a threefold perspective:

1. The content, which is divided into content portions.

2. The logical structure, which is a hierarchy of logical objects.
3. The layout structure, which is a hierarchy of layout objects.

Logical objects are subdivisions of the document based on meaning, and *layout objects* are subdivisions of the document based on appearance.

Content portions are associated with both kinds of objects. For example, a text content portion may be associated, in the logical structure, with a logical object called "glossary definition." After formatting, that content portion may also be associated, in the layout structure, with a layout object called a "block."

In addition, ODA permits the creation of styles, which are collections of attributes affecting the layout and presentation of a document.

In order to represent both the revisable form of a document and the presentation form, ODA provides three classes of document architecture:

1. Formatted
2. Processable
3. Formatted-processable
   (IIA does not currently include the formatted form or the formatted-processable form)

ODA also provides for document classes, where a particular class is defined by a particular generic logical or layout structure. Generic structures serve as templates for the processing of individual documents.

In addition to expressing the logical structure and layout structure at the document level, ODA provides architectures at the content level. ODA currently provides architectures for:

- Text
- Image
- Graphics

It is extensible for other types. This inclusiveness makes ODA useful for standardized interchange of both document-level information and content-level information.

4.2.1 General Concept of ODA

Purpose of ODA

The purpose of the document architecture is to facilitate the interchange of documents in a manner such that:

- Different types of content, including text, image, graphic and sound, can coexist within a document.
- The intentions of a document originator with respect to editing, formatting and presentation can be communicated most effectively.

The Office Document Architecture provides for the representation of documents in three forms:

1. Formatted form
   allows documents to be presented as intended by the originator.
2. Processable form

allows documents to be edited and formatted.

3. Formatted processable form
   allows documents to be presented as well as edited and reformatted.

Alternative terms commonly used are *final form* and *image form formatted form*, and *revisable form* for *processable form*.

Each of these forms allows the originator to express intentions regarding the structuring and/or formatting of the interchanged document.

Overall Concept of ODA

The concept of ODA is based on:

- The existence of a layout view and a logical view of the document, the view from the physical viewpoint (for example, a collection of pages), and the view in the sense of its abstract components (for example, an assembly of sentences).
- The existence of a specific structure and a generic structure. The specific "document" structure is the one that the user may read; the generic structure is the template that guides the creation of the document and that could be reused for its amendment
- The existence of document classes:
  a document class is the set of generic features that are common to a category of documents (for example, Sales Report Form).

Logical Structure and Layout Structure

The key concept in the document architecture is that of structure. Document structure is the division and repeated subdivision of the content of a document into increasingly smaller parts. The parts are called objects. The structure has the form of a tree.

The document architecture permits two structures to be applied to a document:

1. A logical structure
2. A layout structure

Any one or both structures may be applied to a given document.

In the logical structure, the document is divided and subdivided on the basis of the meaning. Examples of logical objects are chapters, sections, figures, and paragraphs.

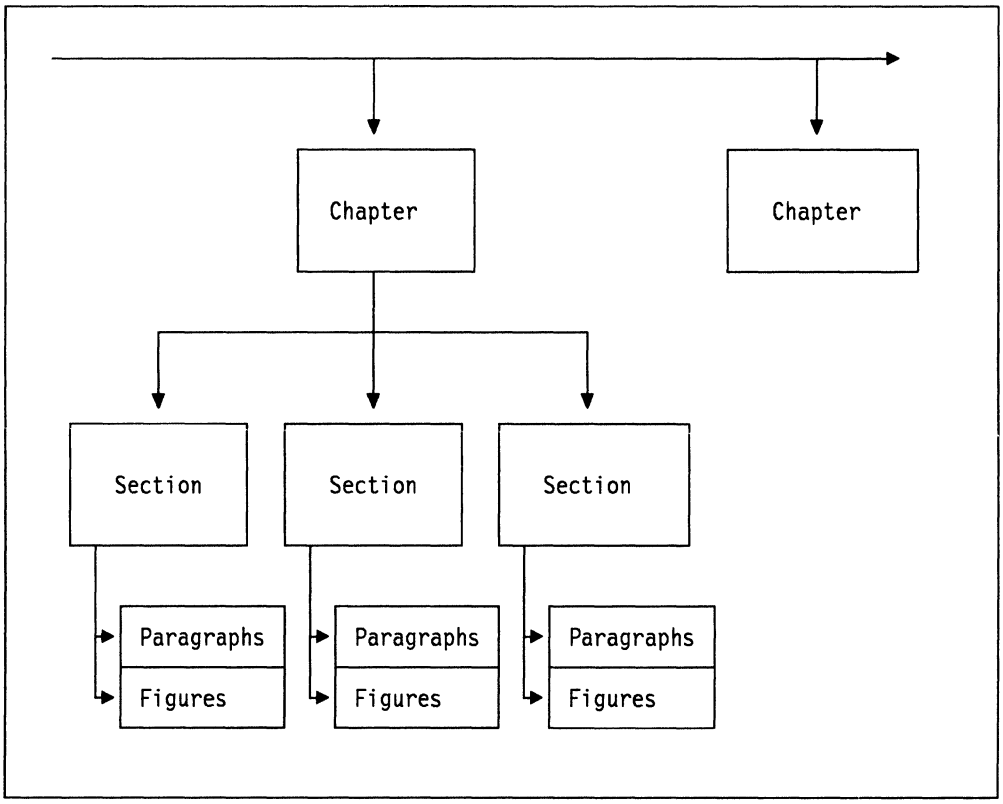


Figure 6. Diagram of a Logical Structure of a Document. The logical structure of a document is determined by the author of the document. It is based upon chapters and the different sections within chapters.

In the layout structure, the document is divided and subdivided on the basis of the layout. Examples of layout objects are pages and blocks.

The logical structure and the layout structure provide alternative but complementary views of the same document. For example, a book can be regarded as consisting of chapters containing figures and paragraphs, or alternatively, as consisting of pages that contain text blocks and/or graphic blocks.

Content Portions: The basic elements of the content of a document are called content elements. For content consisting of character text, the content elements are characters. In the case of images or graphics, the content elements are picture elements (also called pels).

Content Architectures: A content portion associated with a basic logical object or a basic layout object may have a more detailed internal structure. The rules governing such an internal structure depend on the type of content and are called a content architecture. The content of a basic logical object or a basic layout object is structured according to only one content architecture.

Attributes: An attribute is a property of a document, or of a document constituent. It expresses a characteristic of the document component concerned, or a relationship with one or more documents or document components.

Relations Between Logical Structure and Layout Structure: The logical structure and the layout structure are, in principle, independent of each other. The logical structure of a document is determined by the author and

embedded in the document during the editing process. The layout structure is usually determined by a formatting process. The formatting process may be controlled by attributes called layout directives associated with the logical structure.

Specific and Generic Structures: In a document, the logical objects and/or the layout objects can often be classified into groups of similar objects. Therefore the concept of object class is introduced.

The similarity may be related to logical features such as chapter, section, or paragraph hierarchy, to layout features such as size or styl, or to content such as page headers and footings. Even an entire document may be a member of a group of similar documents, a letter, a memorandum, or a report.

Document Profile: The document profile consists of a set of attributes associated with a document as a whole. In addition to reference information such as title, date, and author's name, which facilitates storage and retrieval of the document, the document profile contains a summary of the document architecture features that are used in the document, in order that a recipient can easily determine which capabilities are required for processing or imaging the document.

Generic-document: A generic-document consisting of a document profile and generic structures can be used to assist in the processing of interchanged documents. A generic-document may be interchanged.

For a complete description of ODA refer to *Information Interchange Architecture: Concepts, GG24-3503.*

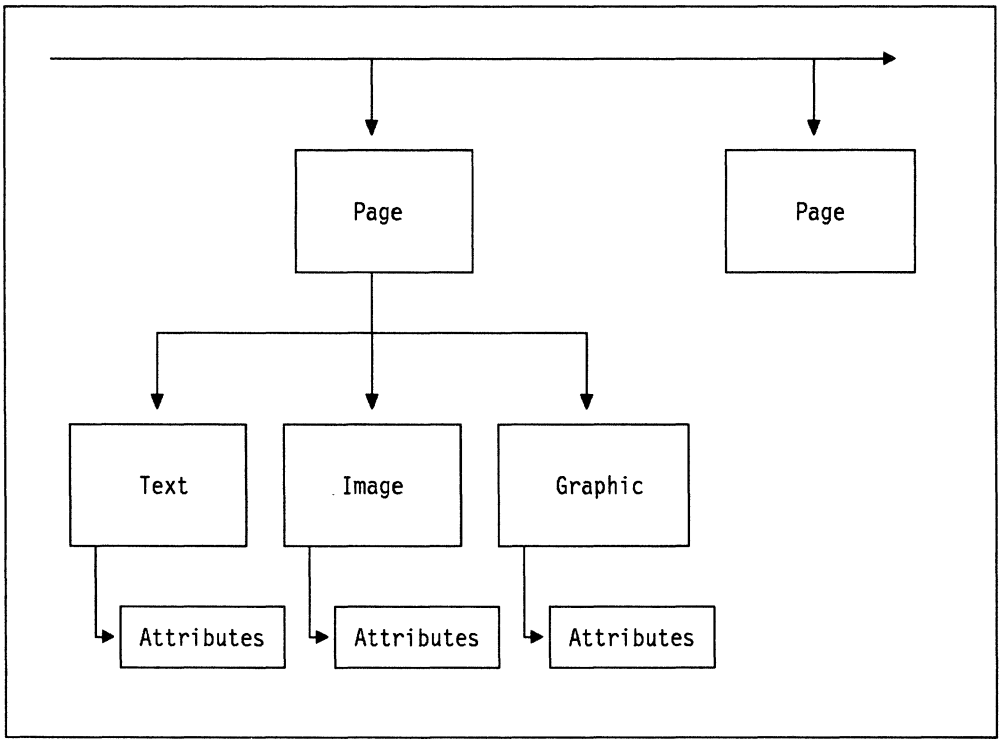


Figure 7. Diagram of a Layout Structure of a Document. The layout structure of a document is determined by the formatting process and controlled by attributes. It is based upon pages or block.

4.3 Industry Standard and non-IBM Proprietary Architecture

In this book we deal with "Industry Standard" and non-IBM proprietary machine readable documentation at the data stream or object level. Our concern is to describe for the benefit of IBM system users how to make data in these formats fit into their systems and documentation.

There are no architectures that are "Industry Standard" in any case; the only architectures widely accepted in the computer industry in this sense are the ISO standards which are introduced in the previous section.

Chapter 5. IBM Data Streams

A data stream is a continuous ordered stream of data elements conforming to a given format. The data streams that are part of SNA Common Communications Support of SAA (CCS) can transfer data between:

- Application programs
- An application program and a printer
- A workstation and an application program

The application programs, workstation, and printer can be on the same system or on different systems.

The data streams in Common Communications Support include:

- The *Intelligent Printer Data Stream* (IPDS*)* is the system-to-printer data stream for all-points-addressable printing.
- The *3270 Data Stream* defines a formatted data stream used to transmit data between an application program and a nonprogrammable workstation or printer.
- The *Character Data Representation Architecture (CDRA)* is the data stream that preserves the original meaning of graphic characters during accessing, processing, conversion, and interchange of character data between and across SAA systems.
- The *Mixed Object Document Content Architecture (MO:DCA)* defines the data stream used to transmit objects from an application program to a programmable workstation or to another application program.
- The *Revisable-Form-Text Document Content Architecture (RFT:DCA)* (RFT:DCA) defines the data stream used to transmit revisable-form text and non-text objects between application programs in an office environment.

5.1 Device Dependent Data Streams

The next sections describe different data streams that are device dependent. The data streams include commands that can be interpreted correctly only by a device that has support for the data stream concerned.

5.1.1 Intelligent Printer Data Stream

The Intelligent Printer Data Stream (IPDS) is used to send data created by the AFP printer driver PSF or an application program to an all-points-addressable printer. The Intelligent Printer Data Stream can carry the BCOCA*, FOCA, GOCA, IOCA, and PTOCA document objects, described in 8.1, "Objects." Therefore, it is possible to print pages containing a mixture of different data types. Different application programs can create source data (text, graphics, image, and bar code) independently of each other. IPDS architecture allows the output of these independent application programs to be merged when printed, so that an integrated mixed-object page results.

Because IPDS architecture is independent of the communication protocol, the same data stream can be transmitted to printers that are attached to channels, to controllers, or to local area networks or any other communication link that allows transparent transmission of data. All data and commands are transferred

through self-identifying structured fields that describe the presentation of one or more pages. IPDS products must be part of the printing subsystem of each environment in which IPDS data streams will be interchanged. All printing subsystems have the following elements in common:

- **Application programs** generate the source data to be printed. Some application programs generate text data that previously would have been directed to line printers. Other application programs generate all-points-addressable text or other data types such as image, graphics, and bar code for IPDS printers.
- **Presentation services** accept source data, transform it into an Intelligent Printer Data Stream without changing the existing source data, and communicate with an IPDS printer. Presentation services also permit the output of line-printer application programs to be enhanced by IPDS capabilities, such as duplexing, overlays (electronic forms), and multiple high-quality fonts.
- **IPDS printers** accept the Intelligent Printer Data Stream. They can attach to several different system or subsystem environments using one or more communication protocols.

The IPDS implementation requirements differ for presentation services products and printer products. For more information, see 5.1.1.2, "Minimum Functions Required for IPDS Support" on page 32.

The detailed reference to the AFPDS data stream itself, to the level of the structured fields and object descriptions is *AFP Data Stream Reference, S544-3202*, to which the reader is referred for that level of technical detail.

5.1.1.1 IPDS Functional Areas

The IPDS architecture is divided into several functional areas, called command sets, each representing a major printer capability. A *command set* consists of:

- IPDS commands, including semantics (the relationship of the command symbol to its meaning)
- Syntax (the command structure and format)
- Architecturally valid values for each field in the command

In addition, the architecture contains a registry of exception-reporting codes for error conditions in each of its command sets and for printer-related failure, fault, or host-notification conditions.

Each command set is further divided into at least one subset of defined function and a subset of optional function. Some command sets have more than one subset of defined function. Command sets that are defined to carry object data also define a data tower that describes the data carried in the "Write" command of the corresponding IPDS command set.

The command-set design allows IPDS architecture to support a wide range of printer products. Product developers can match command-set implementations to the specific needs of their products.

Device Control This command set is composed of the IPDS commands that initialize the environment for a page, communicate device controls, and manage the printer acknowledgment protocol.

Text This command set is composed of the IPDS commands for presenting text information on a page, a page segment, or an overlay.

IO Image This command set is composed of the IPDS commands for presenting images on a page, a page segment, or an overlay.

Graphics This command set is composed of the IPDS commands for presenting graphics on a page, a page segment, or an overlay.

Bar Code This command set is composed of the IPDS commands for presenting machine-readable bar code information on a page, a page segment, or an overlay.

Page Segment This command set is composed of the IPDS commands to store and present IPDS constructs containing text, graphics, image, and bar code information. These stored constructs, which can be merged with a logical page to assume the current environment, are called page segments.

Overlay This command set is composed of the IPDS commands to store and present IPDS constructs containing text, graphics, image, and bar code information. These stored constructs, which specify their own environment and are often used as electronic forms, are called overlays.

Loaded Font This command set is composed of the IPDS commands to load and delete font information.

For some IPDS command sets, a data tower exists that consists of the data carried in the "Write" command of the corresponding IPDS command set. A data tower can be divided into levels. In that case, a higher level of a data tower consists of all lower levels, plus some set of additional functions.

Some data towers are defined and controlled by Object Content Architectures and are simply registered by IPDS architecture. The following data-tower definitions include the name of the architecture that defines and controls each of the data towers.

Text This data tower is composed of Presentation Text Object Content Architecture (PTOCA) control sequences contained in the data field of the Write Text command. These control sequences are required to present text information in a page, a page segment, or an overlay. The text data tower contains two presentation text (PT) levels—PT1 and PT2—defined by PTOCA.

IO Image This data tower is composed of Image Object Content Architecture (IOCA) self-defining fields contained in the data field of the "Write Image 2" command. These self-defining fields are required to present image data in a page, a page segment, or an overlay. The IO-image data tower contains one level—FS10— defined by IOCA.

Graphics This data tower is composed of Graphics Object Content Architecture (GOCA) drawing orders contained in the data field of the "Write Graphics" command. These drawing orders are required to present graphics in a page, a page segment, or an overlay. The graphics data tower contains one level—DR/2V0—defined by GOCA.

Bar Code This data tower is composed of Bar Code parameters contained in the data field of the "Write Bar Code" command. These parameters are required to present machine-readable bar-code information in a

page, a page segment, or an overlay. The Bar Code data tower contains one level—BCD1— defined by the BCOCA architecture.

5.1.1.2 Minimum Functions Required for IPDS Support

To claim support of the IPDS architecture, an IPDS printer product must do the following:

- Implement the DC1 subset of the device-control command set.
- Implement at least one of the following subsets of the IPDS command sets:
  - Text (TX1)
  - Image (IO1)
  - Graphics (GR1)
  - Bar codes (BC1)

To claim support of the IPDS architecture, a presentation services product must do the following:

- For all commands generated by the presentation services, the command must conform to the IPDS state diagram.
- For all commands generated by the presentation services, the command syntax must conform to the syntax defined by the IPDS architecture.

5.1.1.3 Command-Set Support Requirements

To claim support of the text, graphics, IO-image, or bar-code command sets of the IPDS architecture, an IPDS printer product must implement an architecturally defined subset of the command set. Printers can support additional, optional elements of the command set. In addition, a printer product must also implement a level of the corresponding data tower.

To claim support of any other IPDS command set, a printer product must implement an architecturally defined subset of the command set. Printers can support additional, optional elements of the command set.

Refer to *Intelligent Printer Data Stream Reference, S544-3417,* for additional information on IPDS command sets.

5.1.1.4 Data Tower Support Requirements

To claim support of a data tower, a printer product must implement an architecturally defined level of the data tower.

5.1.2 3270 Data Stream

The 3270 Data Stream is a formatted data stream used to transmit data between an application program and a 3270-type workstation or printer. The 3270 Data Stream is based upon the presence of a mapped character buffer in the 3270 workstation. A fixed one-to-one relationship exists between each character-storage location in the buffer and each character position on the display.

An application program uses one of two methods to communicate with the user at a workstation:

- The application program leaves the display surface unformatted and the user uses it in a free-form manner.
- The application program either completely or partially formats the display surface (arranges it into fields) and the user enters data into the fields.

The 3270 Data Stream allows the application programmer to divide the display surface into one active area and, optionally, one or more reference areas; each area is called a *partition*. The partition that is active contains a cursor and is the only partition in which the user can enter data or requests.

5.1.2.1 Minimum Functions Required for 3270 Data Stream

The 3270 functions required for Common Communications Support are called *extended function base support (EBASE)*. EBASE specifies functions in the following categories:

- Query replies
- Structured fields
- Basic 3270 commands
- Basic 3270 orders
- 3270 controls/special characters

Please refer also to 5.1.5, "Line Printer Data" on page 36.

The document *IBM 3270 Data Stream Programmer's Reference* identifies the specific functions required in an SAA 3270 Data Stream.

5.1.3 Character Data Representation Architecture

Graphic characters such as letters of the alphabet and punctuation marks are the fundamental basis of written communication. For processing and communication by computers, they are encoded, processed, and stored as binary numbers, commonly referred to as *code points*. The result of the encoding process for a graphic character set is called a *coded graphic character set*, or simply a *code page*, which associates graphic characters with their corresponding code points.

Over the years, industry, national, and international standards have been created to provide rules for encoding graphic characters. Today it is common place for large communication networks to span several countries. Many problems have resulted from the use of different character representation codes. Some of the more common problems follow:

- The dollar symbol ($) is sent in an unarchitected data file from a US-based host system to a UK system, where it then appears as a pound sterling symbol (£).
- The Personal Computer supports a larger character set than can be processed in a non-PC environment.
- Code point conversion tables vary among different products, producing inconsistent results.

Character Data Representation Architecture (CDRA) is the CCS data stream that defines a set of identifiers, services, supporting resources, and conventions to achieve consistent representation, processing, and interchange of graphic character data[1] in SAA environments.[2]

---

1 Graphic character data is not to be confused with the "graphic" data type, which is used to represent double-byte data in some programming languages.

2 Though CDRA is primarily focused on SAA environments, it is applicable in non-SAA environments.

CDRA supports the principle of data and application independence, which is a vital requirement of distributed computing environments. It follows the premise that the elements of character representation are essentially data attributes, and these elements must be uniquely captured by a method of identification that can be associated with the data. Using CDRA, applications or devices handling data have one identifier from which they can derive all the information needed to correctly identify the graphic characters represented by code points. Having one identifier permits consistent and correct handling of graphic character data.

CDRA defines:

- An *identification mechanism* and associated supporting resources
- Functions such as *tagging* to associate the identifier with data and applications
- Tables necessary for consistent *code point conversion* of coded graphic character data
- Recommended *strategic coded graphic character sets*

These CDRA features are described in the following sections.

5.1.3.1 Identification Mechanism

The CDRA *identification mechanism* is used to specify the identity of graphic character data and uniquely refer to this data at any place in a system. It can be thought of as providing additional information to eliminate the ambiguity inherent in the binary code point. This identification mechanism has two forms:

- The *short form* is a fixed-length two-byte identifier, known as the Coded Character Set Identifier (CCSID), which represents the elements of the long-form identifier.
- The *long form* is a variable-length identifier, composed of multiple elements:
  - Encoding scheme identifier
  - Character set identifier
  - Code page identifier
  - Additional elements as needed for each additional pair of character set and code page identifiers
  - Additional coding-related required information (ACRI), which can be used to specify the ranges of valid first bytes for double-byte characters as implemented by the IBM Personal Computer

5.1.3.2 Tagging

Products implementing CDRA associate the CDRA identifiers with the data objects they manage. This tagging of data objects is the method used to identify the meaning of the coded graphic characters in the object. This allows a graphic character that has different code points assigned in different machine types to maintain its meaning.

The tag may be in a data structure that is logically associated with the data object (known as *explicit tagging*). Alternatively, it may be inherited from the tag fields associated with other related data objects, or from the computing environment itself (known as *implicit tagging*).

5.1.3.3 Code Point Conversion

The process of managing character data representations that are different from the representations expected by applications, devices, or interchange environments is known as *difference management*. This process involves the ability to recognize if a difference exists, and the ability to deal with differences consistently and correctly. CDRA describes how to manage the representational differences in coded graphic characters, and the criteria to be used for the creation of character conversion tables and methods.

5.1.3.4 Strategic Coded Graphic Character Sets

CDRA defines the SAA character sets and code pages that can be used to minimize differences in coded graphic character representations and related potential data loss. When data is transferred between environments using different character representations, the integrity of characters that are common to the two sets of coded graphic characters can be maintained. *Integrity* in CDRA terms is the ability to preserve the meaning of a coded graphic character as defined by the CDRA identifier.

CDRA defines CCSIDs for coded character sets. The two types of CCSIDs are:

- SAA
- Migration and coexistence

The SAA CCSIDs provide strategic direction for new applications by enabling the preservation of character data integrity within a country or a group of countries that use the SAA character set composing the SAA CCSIDs.

5.1.3.5 Minimum Functions Required for CDRA

Products implementing CDRA Level 1 must implement the appropriate functions from the following list:

- The identification mechanism and associated supporting resources
- The tagging function to associate the identifier with data and applications
- The difference management function to provide consistent code point conversion of coded graphic character data
- The strategic coded graphic character sets. This support might entail migrating existing support for current coded graphic character sets.

Additional information about CDRA is in *Character Data Representation Architecture—Level 1 Reference, SC09-1390*. Other books related to CDRA are listed in Appendix E, Bibliography.

5.1.4 SCS

The SNA Character Stream is a sequential character string composed of EBCDIC controls and user data. The primary function of the control codes is to format data either on a printer or a display. SCS control codes can be intermixed with graphic data characters. SCS functions do not include data flow control functions.

Examples of SNA Character Stream control are:

Backspace
Carrier return
Form feed
Horizontal tab

Presentation position
Word underscore

5.1.5 Line Printer Data

Normally applications are programmed to generate output for line printers. This output is often in fixed length records, with each letter is already on the offset where it will be printed.

The first position is used either as an ANSI carriage control character or as a machine carriage control character.

This data stream is often also called the 1403 data stream, because that was the first printer that this data stream was defined for.

The line printer data stream is also suitable for AFP Printing without any need for changing existing application programs. AFP supports this by providing tools to define print definitions externally to application programs, and allowing invocation of them at print time.

Please refer also to 5.1.2, "3270 Data Stream" on page 32.

5.1.6 IBM Personal Printer Data Stream

It's stretching things a little to call this a data stream, as it is principally text based, but it can contain bitmap data, so it does just qualify. Also, it is *known as* a data stream, so it would be confusing for us to treat it as a data object.

PPDS was previously known also as IBM PC ASCII Printer Data Stream, and was designed to handle printing on the IBM PC in an eight bit byte ASCII environment.

PPDS is an expandable standard, and generally devices should ignore unsupported control codes; however, this may not be true of some older devices.

PPDS is essentially text or bitmap interleaved and managed by control codes. There are three types of control codes:

Single byte control codes
Escape sequences
Control sequences functions

PPDS is still found wherever there are PCs, and is probably still the most widely used workstation print data stream.

PPDS is split into three levels, each indicating a level of development:

Level 1 contains support for the following:

- 9 pin printers
- 24 pin printers
- Basic paper handling (new page)
- Limited font selection
- Image Graphics with multiple resolutions

Level 2 contains additional support to level 1 for the following:

- Quietwriter*
- Quickwriter III
- Better font selection

- Better font control
- Enhanced paper handling (cut sheet)
- Better text justification and formatting

Level 3 contains additional support to level 1 and 2 for the following:

- Support for page printers (IBM 4019)
- Random placement of text and graphic images
- Limited drawing by printer
- Selection of paper orientation
- User of typographic fonts

The following single byte controls are supported by most IBM personal Printers:

Table 2. PPDS Single Byte Control Codes

Mnemonic	Decimal	Hex	Function
NUL	00	00	Null
BEL	07	07	Bell
BS	08	08	Backspace
HT	09	09	Horizontal Tab
LF	10	0A	Line Feed
VT	11	0B	Vertical Tab
FF	12	0C	Form Feed
CR	13	0D	Carriage Return
SO	14	0E	Shift Out - Double wide print line mode
SI	15	0E	Shift In - Condensed print
DC1	17	11	Device Control 1 - Select/XON
DC2	18	12	Device Control 2 - Set 10 cpi
DC3	19	13	Device Control 3 - Deselect/XOFF
DC4	20	21	Device Control 4 - Cancel Double wide
CAN	24	18	Cancel (Clear printer buffer)
ESC	27	1B	Escape

Note: This level of detail doesn't really belong in this book, but the authors have lost count of the numbers of times they have needed these and haven't been able to find them.

5.1.6.1 Escape Sequences

Escape sequence functions consist of the Escape character (decimal 27) followed by a code, and in some cases then by parameters.

Table 3 (Page 1 of 2). Common PPDS Escape Sequences

Hex Code	Function	Parameter
X'2D'	Auto Underscore	X'00' off, or X'01' on
X'30'	Set line spacing 1/8 inch	
X'31'	Set line spacing 7/72 inch	
X'32'	Set line space to last Esc X'1B41' value, or to 1/6 inch if none	

Table 3 (Page 2 of 2). Common PPDS Escape Sequences		
Hex Code	**Function**	**Parameter**
X'33'	Set graphic line spacing	Hex number of 1/216 inch increments
X'34'	Set current position as top of form	
X'35'	Set Auto Line Feed	X'00' off or X'01' on
X'36'	Select PC Character Set 2	
X'37'	Select PC Character Set 1	
X'3A'	Set pitch 12 cpi	
X'41'	Set Text Line Spacing	Hex number of 1/72 inch increments
X'42'	Set Vertical Tab stops	n hex numbers specifying n tab stops in numbers of lines, x'00' terminating
X'43'	Set Page length	Hex number of lines, or x'00' and hex length in inches
X'44'	Set Horizontal Tab stops	n hex numbers specifying n tab stops in no. of columns, x'00' terminating
X'45'	Begin Emphasized Print	
X'46'	End Emphasized Print	
X'47'	Begin Double Strike Print	
X'48'	End Double Strike Print	
X'4A'	Relative Move Base Line	Hex number of 1/216 inch vertical move
X'4B'	Normal density bit image (60 dpi, 72 dpi vertical)	First byte is low byte, second high, of two byte count of image bytes; followed by n image bytes. Each image byte represents 8 vertical dots.
X'4C'	Dual Density bit image (120 dpi, 72 dpi vertical)	As for 4B
X'4E'	Set skip perforation	Hex number of lines to be skipped at bottom of page
X'4F'	Reset skip perforation	
X'52'	Set default Tab racks	
X'53'	Begin sub or superscript	X'00' superscript, X'01' subscript
X'54'	End sub or superscript	
X'55'	Set print direction	X'00' bidirectional, X'01' L2R, X'02' R2L
X'5C'	Print All Characters	First byte is low byte, second high, of two byte count of bytes to be printed as graphic characters, including those normally controls
X'5E'	Print Single character	Next character to be interpreted as graphic
X'5F'	Continuous overscore mode	X'00' off, X'01' on
X'64'	Relative move inline forward	Move (byte1 + byte2*256)/120 inches right
X'65'	Relative move inline back	Move (byte1 + byte2*256)/120 inches left
X'6A'	Stop printing	
X'6B'	Set Portrait orientation	
X'6C'	Set Landscape orientation	
X'6E'	Select aspect ratio	X'00', X'01', or X'03'

5.1.6.2 Control Sequence Functions

Control sequence functions form more sophisticated controls for newer printers. They consist of an escape character, followed by a hexadecimal '5B', followed by a one-byte code, followed by a two-byte count (least significant byte first) of the number of bytes following in the sequence.

```
Esc [ Code Count_low Count_high (Count)bytes_of_parameter_data
```

5.1.7 CDPDS

Composed Document Printer Data Stream is a final form data stream created when documents are to be printed.

It is basically a subset of the MO:DCA-P architecture. This data stream contains device dependent controls (medium related fields).

When working with this data stream the device driver is normally GDDM. The software generating this data stream are normally of the DisplayWrite* family.

5.2 Device Independent Data Streams

5.2.1 AFPDS

Advanced Function Printing Data Stream is a final form data stream created as a result of a print request originated from various office, publishing, and business applications.

AFPDS supports a Superset of MO:DCA data stream functions.

It is the input data stream to PSF (Print Services Facility is the Printer Driver for AFP printers) and to GDDM as well as products working with GDDM* (such as GQDF or BrowseMaster) for display, manipulation, and/or conversion.

5.2.2 MO:DCA

Mixed Object: Document Content Architecture defines IBM's SAA presentation data stream.

5.2.2.1 MO:DCA-P

Mixed Object: Document Content Architecture *for Presentation*

This data stream contains a collection of various data objects such as text, images, graphics, and other components such as layout structures.

Presentation documents consists of one or more pages of final form data in a format that is ready to be printed or displayed.

5.2.2.2 MO:DCA-R

Mixed Object: Document Content Architecture *for Resources*

Presentation documents contain objects that are intended to be stored in a library for later reference by presentation data streams. A typical example is a page overlay used by ImagePlus data streams.

5.2.2.3 MO:DCA-L

Mixed Object: Document Content Architecture *for Libraries*

Presentation documents contain objects that are intended to be stored in a library. A typical example is a page overlay used by ImagePlus data streams.

5.2.2.4 DCA

Document Content Architecture in the office world describes the form and meaning of the contents of a document. Documents in this format can be interchanged through a network. The text of a document is one of the two forms: REVISABLE and FINAL.

RFTDCA Revisable Form Text/Document Content Architecture specifies how IBM Office systems interchange documents in revisable form and defines the structure of the data stream

The data stream contains the text and fields containing general formatting specifications.

Contents and format in revisable form can be modified by any person that has access to it.

FFTDCA Final Form Text / Document Content Architecture specifies how IBM Office systems interchange documents in final form and defines the structure of the data stream

The data stream contains the text and control codes representing the formatting specifications.

Contents and format in final form cannot be modified. It is for presentation on a display or printer.

RFTXT Revisable Form Text Documents are created on IBM equipment other than Personal Services /370. An example of these devices is the IBM Displaywriter.

Documents in this format that are created on a workstation can be loaded to a system /370 or AS/400* for further processing.

Documents created with Personal Services /370 or Personal Services/400 can be downloaded to a workstation for further processing.

Chapter 6. Industry Standard Data Streams

6.1 Device Independent Data Streams

The following section describes data streams that are not device dependent. These data streams are supported by different devices.

6.1.1 PostScript

We are treating PostScript as a data stream for the purposes of this document because it is capable of containing mixed data types and describing a full composite document.

PostScript objects are generally denoted by the use of a file type or extension of PS or EPS. *LISTPS* is also seen.

6.1.1.1 PostScript and Encapsulated PostScript

EPS stands for Encapsulated PostScript, and can contain anything a PostScript data stream contains. Encapsulated PostScript objects are intended to be included in other composite documents, while PostScript data streams are generally documents in their own right. These composite documents need not be PostScript data stream; many desk-top publishing products use EPS files for their graphic entities. An Encapsulated PostScript file usually doesn't contain *showpage* commands (the PostScript command to print a page), and depends on the document containing it to control the where and on which page it is printed. It also contains *bounding box* data which defines its size and internal coordinate system.

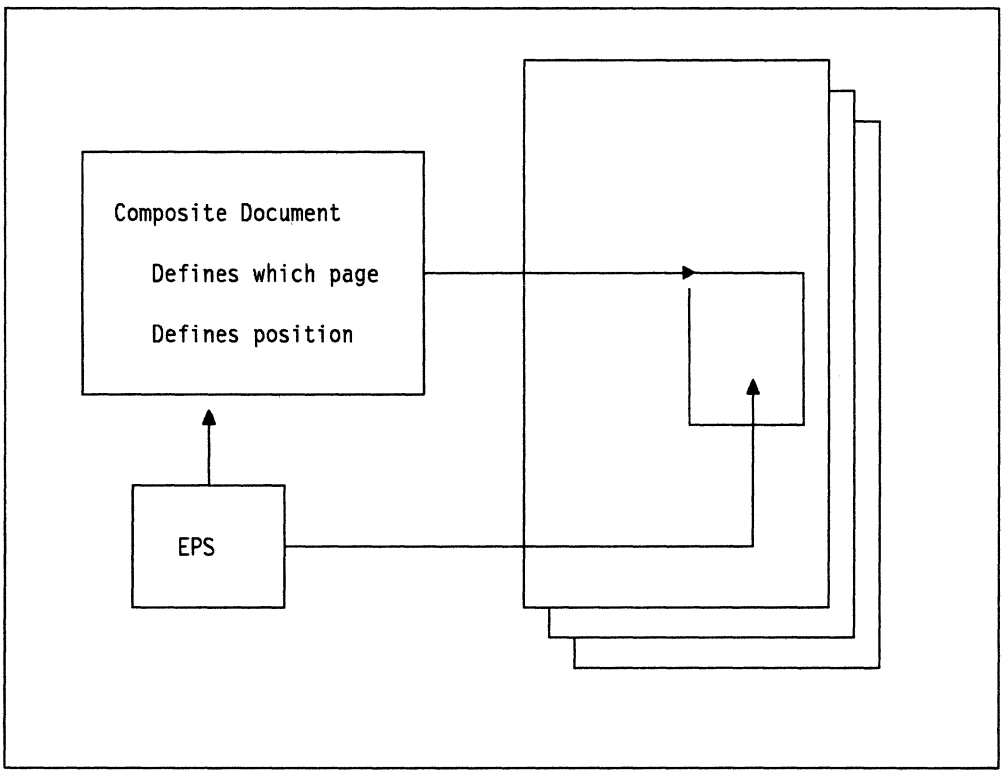


Figure 8. Use of EPS Files

6.1.1.2 Stream Contents

A PostScript data stream can contain text, font data, vector graphics, and bitmap images. While usually monochrome, PostScript is capable of supporting full color.

The PostScript data stream is device and resolution independent, except for image bitmaps, which of course have a defined number of picture elements. Output resolution and color capabilities are dependent on the device at which output is expressed.

Text is stored as characters, and the output form is dependent on the font current at that point. The PostScript data stream can contain the font data, or can just give the name of the font and assume that font data exists at the output device.

PostScript was designed as a *Page Description Language*, and has all the richness of a full programming language. PostScript instances can be very complex (equally, simple PostScript instances can produce excellent output).

6.1.1.3 Usage

PostScript is effectively the industry standard for high quality output in the publishing industry. The vast majority of typesetters now use PostScript as their input medium for both monochrome and high quality color work. Anyone intending to do serious work in the publishing industry must be capable of handling PostScript.

PostScript is also effectively the industry standard for quality output in the workstation world. Low-cost laser printers at the date of publishing are continuing to use proprietary data streams, but all but the cheapest now offer PostScript as an option, and PostScript printers are becoming lower in price. Virtually all desk-top publishing software, and all serious graphics software, offer PostScript as an output option, if not the default and only option.

6.1.1.4 IBM Data Stream Integration with PostScript

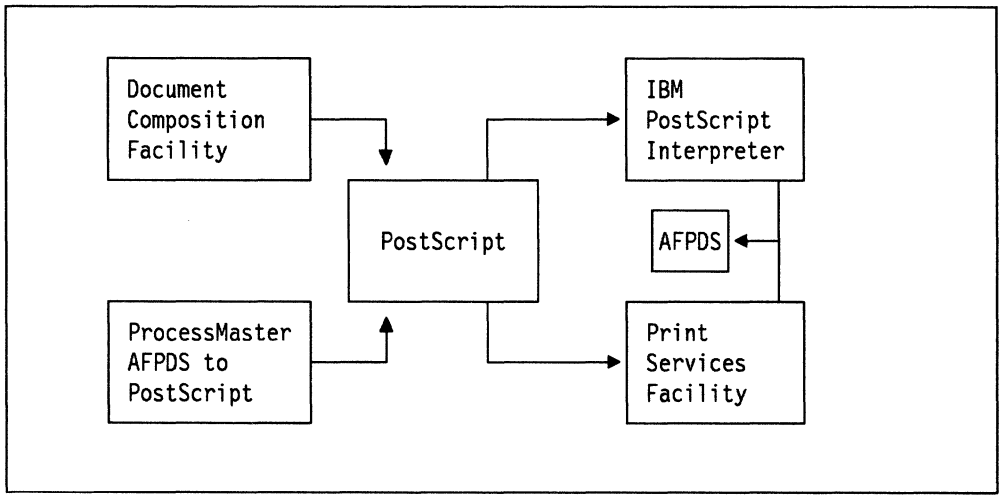


Figure 9. IBM Products and Postscript

IBM's mainframe publishing formatter, DCF, can provide output in PostScript as well as AFP. IBM is in line with the industry. Thus all applications which use DCF as their formatter are PostScript capable.

IBM has AFP as its medium quality output data stream for non-publishing software. We deliberately use the term "medium quality" in this context, because although 240 and 300 pels per inch is regarded as high quality in the computer printer world, in publishing it is at best "medium quality"; a publisher would only start calling output "high quality" above around 1000 pels per inch. The VM and MVS product IBM ProcessMaster contains an AFPDS to PostScript conversion utility, which allows any AFPDS data stream to be used with PostScript devices. Note however, that the files created by this utility tend to be extremely large, even by PostScript standards, and may take a significant time to transfer over communications links and to print.

The reverse process, of converting PostScript to AFP, is carried out by the IBM PostScript Interpreter. This mainframe product allows any PostScript data stream to be output on AFP devices. Note in this case that some PostScript products allow for the inclusion of an "image header" which is often ignored by PostScript devices; the IBM PostScript Interpreter will not, and any files to be used with the IBM PostScript Interpreter should be created *without* this header.

At the time of writing, IBM has announced, but not yet delivered, the capability of handling PostScript as an input data stream to PSF/2 (Print Services Facility/2), in version 1.1. PSF/2 is a product which allows the use of AFP printers in a LAN environment, and, with version 1.1, in a mixed LAN/host environment.

6.1.1.5 Transform Availability

Many programs provide PostScript as output, because of both its capability of providing a device and resolution independent data stream containing text, vector, and bitmap graphics, and also the large range of printers and other output devices that support it.

Fewer programs support it as an input data stream, because of its relative complexity. In general, where it is allowed as input, either only a very simple subset is used, or the input is converted to a bitmap.

6.1.1.6 Device Independence

PostScript defined for a specific printer or operating system can be less than totally device independent. If you have the option (when selecting printer drivers for an application under OS/2* or DOS/Windows, for example) select a *generic* PostScript driver (often just identified as "PostScript Printer"). This will maximize your chances of producing a data stream that can be used anywhere. (Another frequently offered option it is best to avoid is "Image Header" — not all interpreters can handle these.)

This also applies to PostScript when it is being used as a programming language, or writing applications that produce PostScript as an output stream. It is best to ensure that the subset of PostScript that is being written does not contain any printer-specific commands if you want your output to be truly portable. Fortunately this is relatively easy given the richness of the PostScript definition.

Chapter 7. Non-IBM Proprietary Data Streams

7.1 Microsoft Rich Text Format

Though not totally device independent (it contains, for example, font information which may not be appropriate on some output devices — it cannot itself contain font definitions to devices not containing those fonts, like PostScript can), this data stream is not tied to specific devices.

It is essentially a workstation format used to store document information by several Microsoft** application programs. The format documentation is available to developers, and so can be used to exchange information between applications.

Its principal use is as a revisable document definition language for storage and interchange.

An RTF document can contain font and formatting reference information, and character text. Though not sufficiently rich to be described a Page Description Language, the position on page of text, font family, and size can be defined.

An RTF stream can contain embedded picture information; either OS/2 or DOS/Windows metafile, or bitmap. Other formats must be converted before inclusion in the document.

If it is necessary to convert an RTF data stream to another for use by other programs, the Ami Pro word processor can import RTF, and export as many kinds of file format; including RFT:DCA, and a tagged ASCII format that is readily transformed to GML (see A.1, "Conversion Between Ami Pro and GML" on page 147).

7.2 PCL (Printer Control Language)

The PCL Printer Language was and is defined by the Hewlett-Packard** (HP**) Company to drive their range of printers, and has been adopted by several other companies as either the only or an optional way to drive the printers they make as well. This print data stream has evolved as new printers with new functions became available, and the current level is PCL 5.

PCL is a complex data stream, and as a printer data stream does not in general lend itself to ready conversion to other forms. It should not be generated unless it is to be used to drive a printer which understands it.

The PCL data stream allows the printing of:

- Typographic quality text
- Raster images
- Vector graphics

PCL 5 has added the capability of handling scalable fonts to the previous bitmap font capability, and incorporated the HPGL/2 vector graphics definitions into PCL.

The PCL language is defined in detail in *The HP PCL 5 Printer Language Technical Reference Manual, Hewlett-Packard part no. 33459-90903*, which should be referred to for any specific details. We recommend this manual as an exhaustive reference to both PCL and HPGL.

However, to allow simple parsing of the data stream if required, we include a brief overview of its structure below.

Raster graphics included in the data stream may be uncompressed, or compressed using:

- Run length encoding
- The TIF "packbits" coding scheme
- Delta Row compression

The run-length encoding is done with a pair of bytes, the first of which is a repetition of the data in the second byte; a repetition count of 0 says the pattern is not repeated, 1 says it is repeated twice, and so on.

Packbytes (also known as packbits) compression is a form of run-length encoding. Again a control byte precedes the raster data, and determines whether the raster data is to be repeated some number of times or whether the control byte is followed by a number of literal raster bytes. If the sign of the control byte is positive, the control byte is a number of bytes following the control byte that are explicit raster data. If the sign of the control byte (two's complement) is negative, the absolute value of the two's complement number is the number of times the following byte is to be repeated. (Control byte values 0 to 127, add 1 to get number of literal bytes; 129 to 255, subtract from 256 to get the number of repetitions.)

Delta row compression is a scheme that transmits only changes in the bit pattern from row to row. For each row, a control byte contains in bits 7 to 5 the number of bytes to replace, and in bits 4 to 0 the relative offset from the last untreated byte. This is followed by the data bytes. If more than 8 data bytes are required in a row, the command byte is repeated. Offset values of 0 to 30 are from the 1st to 31st byte, 31 indicates that the next byte is to be added to the offset. If this next byte is 255 then additional bytes are to be added until one less than 255 is found.

The PCL language consists of:

- Control codes - essentially the normally defined ASCII control codes
- PCL commands, also known as *escape sequences*
- HPGL/2 commands

PCL escape sequences are either two character or parameterized.

Two-character escape sequences consist of the Escape character (decimal 27, hex 1B) followed by any ASCII character from decimal 48 to 126 inclusive. Parameterized escape sequences have the form:

```
Esc X y val z1 val z2 val z3 . . . val Zn [data]
```

where:

X ASCII 33 to 47 decimal - code indicating parameterized command

y Group type of control - ASCII 96 to 126 decimal

val numeric value (may be preceded by + or -, may contain decimal point). If required by command and value is unspecified, 0 is assumed.

zn Parameter Character (identifies parameter to which previous value applies) - ASCII 96 to 126 decimal

Zn Termination character (identifies parameter to which previous value applies) - ASCII 64 to 94 decimal

data Eight bit data (for example, graphics, fonts). Number of bytes is specified by a value field (usually the last one).

All except X may be optional, depending on the command.

Parameterized escape sequences may be combined, providing the following rules are met:

- The first two characters after the Escape character must be the same in all the commands that will be combined.
- All alphabetic characters within the combined sequence will be lower case, except the last which will be upper case (upper case characters are termination characters).
- The commands are performed left to right.

HPGL/2 commands consist of:

- A two-character mnemonic
- A variable number of parameters separated by a comma, a space, or a sign character
- An optional terminator character, the semicolon

Chapter 8. IBM Object Content Architectures and Definitions

8.1 Objects

Documents and databases can be made up of different kinds of data, such as text, graphics, images, database data, and bar codes. *Object content architectures* describe the structure and content of each type of data that can exist in a document or database.

Formatted data objects are associated with databases, while the following types of objects are associated with documents:

- *Data objects* such as text objects, graphics objects, image objects, and bar code objects
- *Resource objects* such as font objects, which are referred to by data objects.

All objects exist as peers and function as equals. All object content architectures (OCAs) are free to define their own formatting functions. For example, the OCA for text data specifies the spacing between lines and the size of the white space appearing between words.

The object content architectures in Common Communications Support are:

- *Presentation Text Object Content Architecture (PTOCA)* describes presentation-text objects in a document.
- *Image Object Content Architecture (IOCA)* describes image objects in a document.
- *Graphics Object Content Architecture (GOCA)* describes graphic objects in a document.
- *Font Object Content Architecture (FOCA)* defines the structure and content of digital fonts used by data objects in a document.
- *Formatted Data Object Content Architecture (FDOCA)* describes *formatted data*, such as data extracted from a database or read from a file.
- *Bar Code Object Content Architecture* (BCOCA)* describes and generates bar code symbols.

Additional information about object content architectures can be found in *Information Interchange Architecture: Concepts, GG24-3503*. Technical details about the object definitions in the list above can be found in *AFP Data Stream Reference, S544-3202*.

8.1.1 Object Structure

All objects in MO:DCA are made up of two parts: an object descriptor and object data. The content of individual fields varies, depending on the kind of object.

Objects are designed to be carried by, and become part of, a data stream. Data streams are used to pass documents between application programs or between an application program and a device. The data stream carrying the object provides all external relationships for the object.

8.1.2 Presentation Text Object Content Architecture

After text has been processed (formatted), it is in *presentation form*—the text is ready to be presented at a printer. This is the text part of AFPDS. The *Presentation Text Object Content Architecture (PTOCA)* defines presentation-text objects in a document. A *presentation-text object* describes the portion of a text document that has been generated from one of many possible sources such as:

- Output from formatting processes
- Direct generation by processes or application programs
- Transformation from text of different presentation formats.

The presentation-text object space defines the area into which the graphic characters will fit when they are presented. This area has no relationship to the physical media or printed page until the final document is actually created.

8.1.2.1 Minimum Functions Required for PTOCA

PTOCA functions are divided into a PT1 and a PT2 subset. The PT1 subset includes all of the functions required by the most primitive receiver of presentation-text objects. The PT1 subset is the minimum that must be implemented for receivers of presentation-text objects in CCS.

The PT2 subset includes all of the PT1 subset, plus specialized functions such as underscore, overstrike, superscript, and subscript. Detailed information on PTOCA function sets is in *Presentation Text Object Content Architecture Reference, SC31-6803.*

8.1.3 Image Object Content Architecture

Many business applications require the inclusion of image data in documents, such as signatures, logos, media articles, and photographs. The *Image Object Content Architecture (IOCA)* defines the characteristics of image data in a device-independent format, thereby allowing image information to be interchanged among different applications and devices.

Image characteristics that can be represented by IOCA are:

- Image size
- Resolution
- Recording algorithm
- Compression algorithm
- Number of bits per pixel[3]
- Identification of look-up table

8.1.3.1 Minimum Functions Required for IOCA

IOCA function set 10 (FS10) is required for interchanging images in presentation format using the IPDS and MO:DCA data streams. FS10 represents bilevel images which can be compressed using either:

- IBM Modified Modified READ (MMR) compression algorithm
- CCITT T.6 G4 Facsimile compression algorithm

Function set 11 (FS11), a superset of FS10, is required for interchanging images in presentation format using the MO:DCA-P data stream. FS11 represents

[3] A *picture element (pixel* or *pel)* is the smallest element of a displaceable or printable surface that can be independently assigned color and intensity.

bilevel, gray scale, and color images. The following compression algorithms are also supported:

- IBM Modified Modified READ (MMR) compression algorithm
- CCITT T.6 G4 Facsimile compression algorithm
- IBM Adaptive Bilevel Image Compression (ABIC) binary Q-coder compression algorithm
- IBM concatenated ABIC compression algorithm
- ISO/CCITT Joint Photographic Experts Group (JPEG) compression algorithms

Function set 20 (FS20) is required for interchanging images in the library format of the MO:DCA data stream. FS20 can represent up to 24 bits-per-pixel color images.

Detailed information on IOCA function sets is in *Image Object Content Architecture Reference, SC31-6805.*

8.1.4 Graphics Object Content Architecture

The term *computer graphics* refers to the definition and representation of graphic elements used to build pictures for presentation either on hard-copy devices (such as printers and plotters) or on soft-copy devices (such as vector or raster displays). The *Graphics Object Content Architecture (GOCA)* uses primitives and attributes to define the structure of computer graphics; GOCA also defines operations for manipulating these graphics.

8.1.4.1 Structure of Graphic Objects

Segments are the basic units from which a picture is constructed; they are uniquely identified, self-contained collections of primitive drawing orders and attributes. Primitives include things such as:

- Lines and relative lines
- Full arcs, partial arcs, and fillets (rounded corners)
- Character strings
- Areas
- Images

Typically, attributes describe characteristics of primitives; for example:

- Color
- Line attributes such as type (for example, solid) and width
- Character attributes such as precision, angle, character set
- Patterns

Every segment is either chained or nonchained. A collection of one or more chained segments defines the picture to be drawn. A picture can be subdivided into "subpictures." Nonchained segments typically define "subpictures" that are incorporated into the main picture by being called from another segment.

8.1.4.2 Minimum Functions Required for GOCA

The minimum function set required for GOCA interchange is the DR/2V0 function set used in either a presentation MO:DCA data stream or an IPDS data stream. DR/2V0 orders are matched to the capabilities of some typical output-only displays and some printers. The functions include curved lines, areas, and images.

The DR/3V1 function set is required for interchanging graphics pictures in the library format of the MO:DCA data stream. DR/3V1 includes additional functions such as:

- General clipping paths
- Individual primitive attributes
- Extra curve-generating primitives
- Raster operations to support the requirements of sophisticated workstations

Detailed information on GOCA function sets is in *Graphics Object Content Architecture Reference, SC31-6804.*

8.1.5 Font Object Content Architecture

Graphic characters are the visual representation of symbols used in text; they are letters, numerals, punctuations, or any symbols that represent information. A *font* is a set of graphic characters that have a characteristic design, or a font designer's conception of how associated graphic characters should appear. This sentence shows examples of the *italic font*, **bold font**, and SMALL-CAP FONT.

Font Object Content Architecture (FOCA) defines the parameters required to describe digital fonts used by text and graphic editors, document formatters, and presentation devices. FOCA permits product applications, document references, and presentation devices to access font information.

A method of storing fonts at the system level is described in *Host Font Data Stream Reference, S544-3289.* Using this method, font resources can be managed by a system for font referencing and data access.

Methods of referencing fonts at the data stream level are described in the *Mixed Object Document Content Architecture Reference, SC31-6802,* and in *Document Content Architecture: Revisable-Form-Text Reference, GG23-0758.* Using these methods, a document data stream can identify the font resources needed for formatting and presentation.

A method of accessing fonts at the system level is described in the *SAA Common System Programming Interface: Presentation Reference, GG26-4359.* Using this method, a system can determine which font resources it has available and can obtain specific information from those resources.

A method for accessing fonts at the printer level is described in *Intelligent Printer Data Stream Reference, S544-3417.* See 5.1.1, "Intelligent Printer Data Stream" on page 29 for additional information on the IPDS architecture.

8.1.5.1 How Digitized Fonts Are Used

Font resources are made available for text processing by font production, font storage, and font accessing. FOCA provides the common and consistent font information required for text processing. It also supports font production by defining the font attributes and their interdependencies; however, FOCA does not define how that information is to be generated or modified.

FOCA supports font storage and font access by defining the set of font attributes required by the SAA application environments, and by defining a general format for that information. Application programs that implement FOCA must use the defined font-parameter definitions; however, the application programs are free to define their own internal format for that information.

FOCA defines a set of font-referencing parameters, which may be used to specify and describe a font resource. Each implementing product may specify a set of required font resources; the implementing product may also specify the character content of those font resources. The content of font resources is not defined or controlled by FOCA. For consistency when interchanging and presenting documents, all receiving sites, processing application programs, and presentation devices must have access to the same or equivalent font resources.

FOCA supports the presentation process by allowing device-specific techniques of character-shape representation and presentation. FOCA also permits font producers and product implementers to make use of more generic representation techniques.

8.1.5.2 Minimum Functions Required for FOCA

Font Object Content Architecture Reference, S544-3285, lists the font parameters (or attributes) defined in FOCA that must be supported for various levels of font information interchange. An interchanged font need not contain all of the information specified by one of the attribute lists, but the processing application programs must be able to accept, build, or pass through all of the information contained in the supported attribute list without information being lost. To ensure that no information is lost during data interchange, the following attribute lists are required:

- Font descriptive parameters
- Font character set parameters
- Font metric parameters
- Character metric parameters
- Character shape parameters
- Code-page parameters

8.1.6 Formatted Data Object Content Architecture

In interconnected networks, you may need to extract data from, or add data to, a central file or database that can be in a different SAA system. Likewise, you may need to interchange data extracted from a database with an application that is on the same or a different system. Interchange can occur, for instance, when exporting or importing files, or when passing parameters from one application program to another, in the same node or different nodes of a network.

Typically, formatted data[4] comes from, or is intended for, databases[5] or files. The *Formatted Data Object Content Architecture (FD:OCA)* is used to describe data from databases and traditional application programs. FD:OCA can be viewed as a language that makes it possible to express the present format and meaning, as far as is relevant, of any given data item. *Format and meaning* refers to those aspects of the data that are relevant for a program in a given environment, namely what the data type and its representation are. FD:OCA constructs can express such properties and attach them to the data.

---

4 The term *formatted data* refers to (1) traditional data-processing "data" that has a fixed and strict format and (2) any data that has an unarchitected, but known, format and meaning, and needs a corresponding description.

5 In the context of FD:OCA, the term *database* is used to mean small or large data collections, with or without internal structure and interdependencies; in other words, simple files are always included when the term database is used.

8.1.6.1 The Structure of Formatted Data Objects

A formatted data object has two components—a descriptor and a value.

- The *descriptor* describes the format and structure of the value part of the formatted data object. It tells the data type and the representation used for the individual parts, and how together they make up the value.
- The *value* contains the described data.

Except for the constructs defining where the value begins and ends, no other architectural constructs are intermixed with the data. The data occurs as it has been read from the database, or as it would be recorded in the database.

Depending on the interchange purpose, the formatted data objects are embedded in architected constructs of another CCS architecture, such as the Distributed Relational Database* Architecture (DRDA*). The embedding architecture identifies and brackets the formatted data object and its components, as appropriate within its syntax.

8.1.6.2 Minimum Functions Required for FD:OCA

The descriptive facilities of FD:OCA are divided into a base subset and a DRDA support subset. The base subset (subset 0000) functions allow products to perform a basic interchange of formatted data. The DRDA support subset (subset 0100) includes the complete base subset, plus the meta data construct. The meta data construct is used to encode additional application meaning of the data it is associated with. Subset 0100 is required for all products that access data from relational databases built on the Distributed Relational Database Architecture.

Detailed information about FD:OCA is in *Formatted Data Object Content Architecture Reference, SC31-6806*.

8.1.7 Bar Code Object Content Architecture

The Bar Code Object Content Architecture (BCOCA) is used to describe and generate bar code symbols.

A bar code is an accurate, easy, and inexpensive method of data presentation and data entry for Automatic Identification (AutoID) information systems. Bar codes are the predominant AutoID technology used to collect data about any person, place, or thing. Bar codes are used for item tracking, inventory control, time and attendance recording check-in/check-out, order entry, document tracking, monitoring work in progress, controlling access to secure areas, shipping and receiving, warehousing, point-of sale operations, patient care, and other applications.

A bar code is a predetermined pattern of bars and spaces that represent numeric or alphanumeric information in a machine readable form. The way the bars and spaces are arranged is called *symbology*. The Universal Product Code (UPC), the European Article-Numbering (EAN) system, 3-of-9 Code, Interleaved 2-of-5, and Code 128 are some examples of symbologies.

BCOCA can exist in, or be invoked by, a number of environments. Each of these controlling environments can be specialized for a particular application area. For example, the controlling environment can be:

- The environment involved in electronically distributing documents in a network (such as MO:DCA)

- A presentation system communicating with hard copy presentation devices (such as IPDS)

Bar code data objects stored in the BCOCA format are device-independent, and can be presented on any device that supports BCOCA.

8.1.7.1 Minimum Functions Required for BCOCA

BCOCA provides a wide range of bar code function to cover many different symbologies that are defined for a variety of uses. Not all of the defined BCOCA function is supported by all BCOCA receivers.

A subset of the full capabilities of BCOCA, called BCD1, is defined to specify the minimum support required of all BCOCA receivers. Each field within a BCOCA data structure allows a range of possible values and also identifies the values that every receiver supports. Most receivers support more than the minimum ranges.

Detailed information about BCOCA can be found in *Bar Code Object Content Architecture Reference, S544-3766.*

8.2 Font Overview

The information which follows is specific to AFP and IBM applications in general. Some of these terms are generally used in the industry, but such as "code page" and "code point" would not be generally understood.

A *font* refers to one size and one *typeface* in a particular *type family*. An example of a type family might be **Times New Roman****. **Medium**, **Bold**, or **Italic** might be defined typefaces within that family. A *font* is strictly speaking a fully qualified description of a typeface with a defined size — **Times New Roman Italic 12 point**.

Both typefaces and fonts are often copyright of the designer or the employer of the designer, and the names equally so. For example, "Helvetica" is a copyright name, and a font used in a program cannot be called Helvetica** unless it is licensed from the copyright owner. Sometimes type faces become so popular that they are widely copied, and the term "type family" becomes extended to include all the different people's expressions of the same basic design. There are many designs of *Garamond*, for example, from many different type designers; but they all share some of the common characteristics of the original Garamond design.

The size (height) of a font is usually measured in points (1 point = 1/72 inch).

The size (width) of a font is called pitch and is measured in characters per inch.

The family and style will determine the artistic characteristics of the font.

An example of an IBM font might be **Sonoran-Serif 16 point Roman Medium**; a **Roman** style of the **Sonoran-Serif** family, in a **medium** weight and **16 point** size.

Each font is given an ID (Font Global ID (FGID)) and name.

FGID	NAME
3	OCR-B
11	Courier 10
19	OCR-A
244	Courier 5

A graphic character is a letter, a numeric digit, or a symbol. Each graphic character is given a character ID and a description.

Character	ID	Description
A	LA020000	A capital
a	LA010000	a small
é	LE110000	e acute small
È	LE140000	E grave capital
1	ND010000	one

Character Set Set of Characters.
Each character set has a name and ID. For example, character set ID '697' is called 'Country Extended Code Page' and it has upper- and lower-case letters, numeric characters, symbols and several countries' unique characters and currency symbols such as Ã, Æ, Ø, fl, $ ¥, etc.

Code Point A code point is a one-byte binary value representing one of 256 potential characters. Each of the 256 combinations is normally referred to by its bit configuration in hexadecimal, with two hex characters per byte: X'00' to X'FF'.

The AS/400 uses the Extended Binary Coded Decimal Interchange Code (EBCDIC) to represent characters. Printable characters in EBCDIC are restricted to the values of X'40' through X'FF' (192 values). The code points between X'00' and X'3F' are reserved for printer instructions and commands.

Code Page A code page is a mapping table that assigns graphic character IDs for a character set to specific code points.

There are several hundred different characters that could be created for a given character set and only 192 printable character values or code points. A code page is used to determine which characters are to be assigned to the available code points.

Some common code pages are: '00500' for international #5, '00437' for PC code page and so on.

A character set and code page combination is used to specify the group of symbols available for printing and how they can be referenced.

Character Set	Code Page	Country or Name
00043	00013	Netherlands
00697	00500	International
00697	00284	Latin America/Spain

A character set and code page combination is used on the AS/400 as one of the system values QCHRID or as the printer file parameter CHRID. These values can be changed by the user (with adequate authority) with CL commands.

8.2.1 AFP Font Resources - AS/400

In Advanced Function Printing* terminology, a font consists of three parts:

- Font Character Set
- Code Page
- Coded Font

These three pieces are stored on the AS/400 as *FNTRSC (font resource) objects.

Font Character Set Font character Set is similar in concept to the character set listed above, but a font character set defines font properties (size, style, weight) in addition to the set of characters. The naming convention uses a font character set ID instead of a FGID.

For example, font character set id C0S0CR10 is the character set of Courier Roman 10-PT font.

A font character set is an object on the AS/400 with FNTCHRSET attributes and an object type *FNTRSC.

Code Page Code page function is the same as defined earlier.

A code page in AFP has an object attribute CDEPAG in object type *FNTRSC.

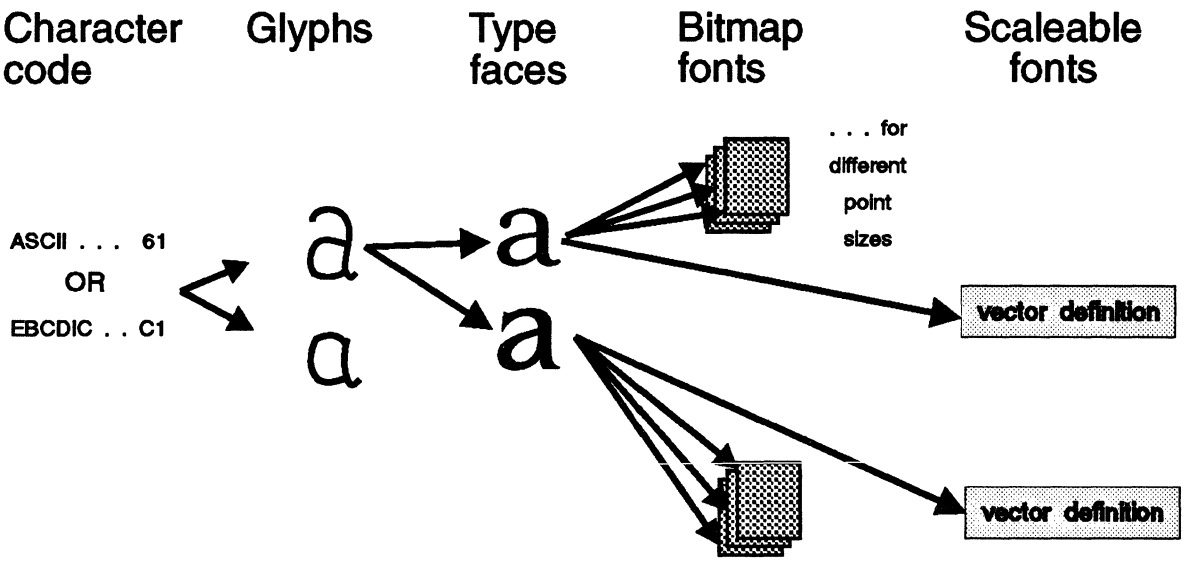
Coded Font A coded font associates a code page and a font character set as a pair. A single-byte coded font contains one code page and one character set pair.

For example, the coded font ID X0GT10 is the association of the font character set C0S0AE10 (APL ROMAN 10-PT) and the code page T1S0AE10 (APL). A coded font has an object attribute of CDEFNT in object type *FNTRSC.

A double-byte coded font contains more than one code page and font character set pair; each pair is called a coded font section. A double-byte coded font requires a two-byte code in text for each graphic character.

- The first byte to identify the section
- The second byte to identify the code point in that section.

Glyph A glyph is a graphic shape which is one of the representations of a character or symbol. A character code defines a given character, given a code page. That character has one or more glyphs which can represent it. A typeface will have a representation of one of those glyphs.



The advent of *scalable fonts* has confused the issue a little. By definition, "scalable" means they don't have a fixed size, so they don't fit the classical definition, unlike *bitmap* fonts, which are representations of a typeface at a specific size. Scalable fonts should really be called scalable *typefaces*, but the usage is too well entrenched now to change.

Chapter 9. Defined Standard Object Content Architectures and Definitions

9.1 Office Document Architecture

Below are definitions for architectures within office document architecture.

9.1.1 CCA

Character Content Architecture contains character data, positional controls and attributes that represent formatted text.

Its purpose is to carry the text part of presentation documents. CCA is an ISO architecture and is part of ODA. It compares to PTOCA within MO:DCA.

9.1.2 RGCA

Raster Graphics Content Architecture contains the raster image data and the attributes that apply to the image.

It is intended to carry raster image information in presentation, revision, and resource documents. RGCA is an ISO architecture and is part of ODA. It compares to the IOCA part of MO:DCA.

9.1.3 GGCA

Geometric Graphics Content Architecture contains the drawing orders, positional controls, and attributes that represent a vector graphic.

Its purpose is to carry vector graphics information in presentation, revision, and resource documents. GOCA is an ISO architecture and is part of ODA. It compares to the GOCA part of MO:DCA.

More information about ODA can be found in the Red Book: *Information Interchange Architecture: Concepts, GG24-3503.*

9.2 Computer Graphics Metafile (CGM)

The CGM format is ANSI standard X3.122-1986.

Data Type: This standard defines a file format capable of storing vector graphic data. It is largely based on the Graphics Kernel System (GKS), an early attempt to define a graphical language callable from high level computer languages.

It is reasonably widely used by workstation based packages for the storage or interchange of vector graphical files.

The standard defines simple graphics primitives which can be used to build up vector graphic pictures:

- Lines and polylines
- Arcs, circles, and ellipses
- Text
- Rectangles and polygons

together with appropriate attributes, including area fill.

This standard does not define any notion of structure, such as graphics segments defined in later standards such as PHIGS; each graphics primitive stands alone. The most that the CGM standard does for grouping information is to define "bundle" information for graphics attributes for different types of primitive; so for a line, reference to a particular line attribute bundle defines the line type, color, and thickness.

The text definition of which it is capable is limited, and many of the packages that use it will code typographical text as vectors and polygons in CGM, to ensure fidelity of representation. It could not be used to define a composite document with typographic text in a revisable form.

Compression: This standard does not define any data compression.

Transform Availability: Many workstation packages import and export this file format. GDDM on the host will also convert this file type to ADMGDF for use with mainframe software (see 13.1.1, "Color Pictures" on page 103). Corel** Draw on the workstation will both import and export this format. However, see the note above concerning the text limitations of this file format.

The CGM standard was defined as the interchange format for business graphics for the CALS initiative of the US Department of Defense, so most defense suppliers will have software to handle the format.

Chapter 10. Industry Standard Object Content Architectures and Definitions

This chapter describes architectures that can be considered to be industry standards.

10.1 Hewlett-Packard Graphics Language

Sometimes also known as "Industry Standard Graphics Language" (ISGL), this is a data format which was originally designed to carry instructions between a program which created graphics, and a plotting device. IBMGL and IBM-GL (IBM Graphics Language) is occasionally seen as a data type identifier; this is functionally identical to HPGL.

HPGL is often thus termed *the* "plot file" format (though there are other plot file formats).

The current level of HPGL defined by the Hewlett-Packard company is HPGL/2 which has extensions over the original language. The HPGL/2 language is defined in detail in *The HP PCL 5 Printer Language Technical Reference Manual, Hewlett-Packard part no. 33459-90903.* HPGL/2 commands can be included in the printer data stream PCL 5 (q.v.).

Besides being an output form for plotters, the IBM 4019 and 4029 Laser Printers also accept this type of data object when running in plotter mode. The resulting images are similar or identical to those produced by many IBM and Hewlett-Packard multipen graphic plotters.

Other plot file formats are not considered in this book, as it is only HPGL that has any significance currently in the printing and publishing world.

Data Type: Vector graphic file type.

The file contains essentially simple graphics primitives and device commands.

Complexity: Though the full definition of the language includes simple text and more complex graphics primitives, many programs restrict themselves to a very simple subset of the language, often no more than "pen up," "pen down," and "move" commands. The format has no structural information, and is not capable of carrying typographic text except as actual graphics orders. Its principal use today is for output of simple business graphics and computer aided design (CAD) drawings; many business charting programs are capable of this format as output, as business charts are often produced on color transparency through plotters.

Transform Availability: Though originally primarily an output format, and still heavily so used, because of the need to incorporate business graphics and CAD output into printed documents and other publishing output, there are many publishing products which will accept this as an input format. Corel Draw will accept HPGL as input, and can be used to modify these graphics before storing them forward for use in a more conventional publishing or printing format. Below is a sample HPGL plot file

```
IN;
PA;
SP1;
PU1512,4606;
PD4170,8670;
PD4745,3707;
PU3128,5469;
PD3162,5408;
PD3199,5347;
PD3237,5286;
PD3277,5226;
PD3318,5165;
PD3361,5104;
PD3405,5043;
PD3449,4982;
PD3494,4920;
PD3540,4859;
PD3586,4797;
PD3632,4735;
PD3678,4673;
PD3724,4610;
PD3770,4548;
PU0,0;SP0;
```

10.2 GIF

GIF is the Graphics Interchange Format defined by CompuServe** Incorporated.

GIF is a popular format for the exchange and interchange of raster data on PC platforms.

Data Type: Raster file type, supporting bilevel, grayscale, and color bitmaps. The grayscale or color is defined by reference to a global or local color table (the GIF format can hold multiple images in a single file, and each may have individual color tables, or a single table may be defined for the file).

Compression: The GIF file type is compressed using a LZW algorithm, which gives good compression ratios. Note, however, that the LZW algorithm is now claimed to fall under patents held by Unisys, so code handling this format should be licensed to use the LZW method.

Complexity: The full specification has many subtypes, many of which are disregarded by some software that claims to handle the format. Much software will only deal with basic GIF, which means that users cannot be certain of full functionality with display, print, or transform software in all cases. Test typical cases before depending on software or files; however, some results will almost always be obtained, though perhaps with inappropriate color or other artifacts. When it is used properly, it can have excellent results.

Transform Availability: Transforms are readily available to and from most other raster filetypes, and from some vector filetypes. Many of these transforms are freeware or shareware. See, however, the notes under "Complexity," and test the transformation software before depending on any of them, particularly when dealing with larger numbers of colors.

10.3 Tagged Image File Format (TIF)

Tagged Image Format objects are generally denoted by the use of a file extension of TIF, occasionally TIFF.

TIF is now a standard capable of defining complex high quality images, as well as simple bilevel bitmaps; it is perhaps most familiar in the latter aspect, but as an established standard, and with the increased demands being placed on computer systems in terms of resolution and color values of images (both numbers of expressible colors and device independence of color), it is the first where it is going to become an even more important interchange format than before.

10.3.1 Content

The TIF standard defines two levels of compliance, *baseline* and *extended*. All programs that create or use the TIF format should be capable of handling anything in the baseline definition, but functions in the extended standard are optional — complying applications need not create or use them. However, applications receiving extended function they can't handle *should* do so gracefully, and where possible make best use of what they can understand. Clearly, some programs will achieve this better than others.

We outline here what capabilities are in version 6.0 of the TIF specification, published by ALDUS. Version 6.0 is dated July 1992; most applications will be written to the previous specification, level 5.0, so we will indicate which facilities are new in version 6.0. The authors of the specification state that if applications written to the previous specification were properly coded, they will be able to use files to the new baseline standard.

10.3.1.1 Types of Files

A baseline TIF file can hold the following types of images:

- Bilevel
- Grayscale
- Palette color
- RGB color

Extended TIF files can hold the following types of images:

- CMYK color (V 6.0)
- YCbCr color (V 6.0)
- CIE L*a*b* (V 6.0)

(Any image types you are unfamiliar with are described briefly in B.4, "Types of Bitmap Images" on page 176.)

10.3.1.2 Compression type

A baseline TIF file can have the following compression types:

- No compression
- Packbytes encoding (a simple run-length encoding - see 7.2, "PCL (Printer Control Language)" on page 45 for an explanation of this scheme)
- CCITT T.3 encoding (modified Huffman)

Note: We have come across some applications that cannot import any compressed TIF files — just because an application says it can handle TIF files doesn't mean it meets the standard. As always, where this is critical, test it out.

Extensions add the following new compression methods:

- CCITT T.4
- CCITT T.6
- LZW compression
- JPEG compression (V 6.0)

CCITT compressions are defined for bilevel images. LZW and JPEG compression are aimed at grayscale and color renditions, though LZW will work on bilevel. JPEG is a family of compression methods defined by the Joint Photographic Experts Group, and contains both lossless and lossy compression techniques. (All other compression methods mentioned are lossless.)

JPEG techniques are targeted at large photographic images requiring high compression ratios, and even lossy JPEG compression should not give significant degradation of the image.

It was originally thought that the LZW (Lempel-Ziv & Welch) algorithm was public domain. It has now been claimed by UNISYS to be covered by a patent of theirs, and UNISYS claim that code expressing it should be licensed by them. Apparently there are also other companies holding patents that might affect code using LZW algorithms.

The upshot of this is that the LZW technique may become less widely used than is currently the case, and as it is an extension to the baseline standard, you should not expect any particular program to handle LZW compression unless it specifically states so.

10.3.1.3 Strips and Tiles

Images tend to be large, so to help practical manipulation of the image, the standard allows for partitioning of the image into manageable chunks (the standard recommends a size of about 8KB as reasonable).

The baseline standard does this by partitioning the image into strips, while version 6.0 now introduces the idea of tiles.

10.3.1.4 Colorimetry

Plain RGB color information is device-specific; a triplet does not define a precise color.

For high quality work, this is not acceptable, and to transfer precise color information for photographic and prepress work, the TIF extensions allow the specification of colorimetric information, which allows mapping of the RGB triplet information onto a defined international color standard (CIE 1931 XYZ). This is largely new in version 6.0.

We can expect to see this aspect, and the device-independent CIELAB colorimetric encoding, become more important at the higher quality, professional end of the market over the next few years.

Chapter 11. Document Languages and Formatting Languages

This chapter deals with document and formatting languages. The definition and description of a document and a formatting language are included. There are also descriptions of several document and formatting languages.

11.1 What is a Document Language?

A document language is a format-, device-, and software-independent data stream which is capable of defining a composite document. Document language instances usually contain the text portion of the document directly, and refer to external files containing the non-text elements, so they are not strictly equivalent to composite document data streams, but they are functionally equivalent.

For example, to include a piece of artwork in a GML document, either the GML Starter Set `.im` macro or the IBM BookMaster `:artwork.` tag is used, both of which refer to the file name of the included graphic.

Normally, SGML behaves in a similar fashion, using entity definitions, but it is possible to define an SGML Document Type Definition to include non-text elements directly inside the document.

A document written in a document language contains information defining the structural elements of the document. The position and boundaries of document elements such as headings, paragraphs, lists, figures, tables, and so on, as well as the elements they are composed of, are precisely defined. As document languages are usually humanly readable, the user can see these definitions, usually called *tags*.

Because they contain this information about the structure and meaning of the elements of a document, document languages are usually considered to be a level above data streams. A document language expression of a document contains more information about the document than a revisable format expression which just contains the text, graphics, and layout, font, and formatting information.

Because a document language is format independent, format information must be added, usually by a program called a "formatter." The formatter program adds format information dependent on the structural element and format instructions contained in a *style definition*. For example, an element may be defined as a chapter heading, and the style definition may instruct the formatter to print chapter headings at the top of a new page, in a specific font at a defined size, and leave so much whitespace afterwards.

The output from the formatter is usually a formatted data stream.

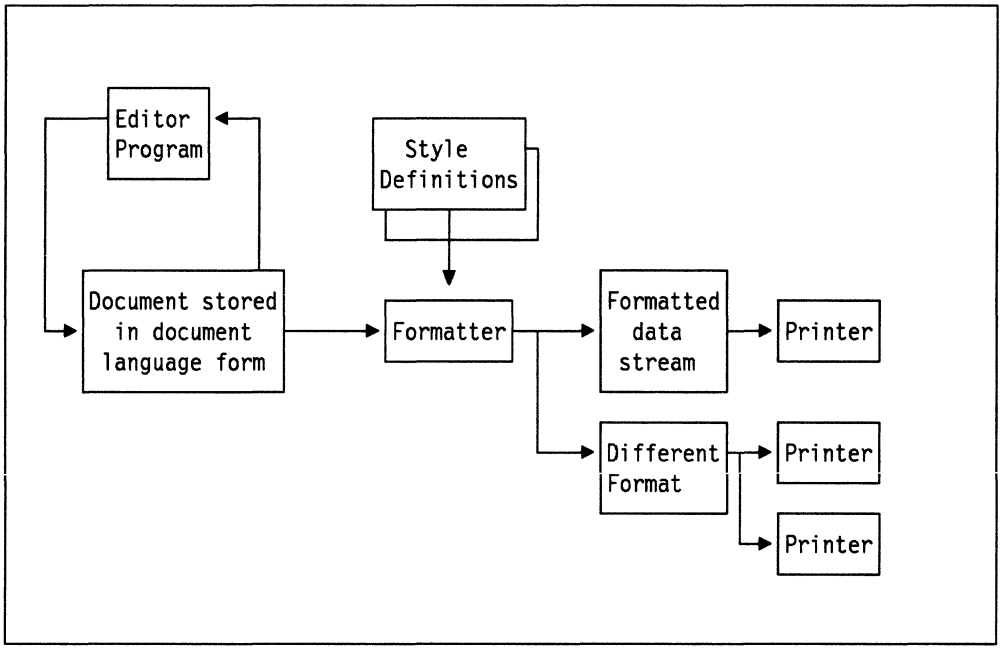


Figure 10. Document Language Processing

The reverse process to formatting, creating document language from formatted documents, is difficult; usually this can only be done by making assumptions drawn from the format, which are not always correct. This reverse process almost always involves human checking or editing.

11.2 What is a Formatting Language?

A formatting language is very similar to a document language, except that it does not usually define structural information; the control tags define directly how the output is to be formatted by the formatter program, and usually no style definition is involved.

Formatting languages have largely been superseded by document languages and integrated publishing or word processing programs.

While not considered in detail here, the file formats stored by workstation based desk-top publishing programs and the more sophisticated word processing programs can be considered to be proprietary document languages, formatting languages, or data streams; which one they are depends on how the program works.

Some store documents in a form similar to a document language; an example is Ventura Publisher**, which stores its documents using a tag language (though it does store some formatting instructions as well).

Some store text and formatting instructions, and can be considered to be proprietary formatting languages, because they must be run through the formatting portion of the package before they can be sent to any output device in a usable form. Very few store in a printable data stream.

11.3 Examples Considered Here

The document languages considered in this document are:

- GML Starter Set (IBM DCF Starter Set GML)
- IBM BookMaster
- SGML (Standard Generalized Markup Language)
- LaTeX
- XICS (Xerox Integrated Composition System)

The IBM document languages IBM DCF Starter Set GML and IBM BookMaster grew out of a formatting language called SCRIPT, and LaTeX is based on the formatting language TeX.

Formatting languages not associated with specific editing programs are usually now more of historical interest than actual use, but three are briefly considered in this book,

- SCRIPT
- TeX
- nroff and troff

SCRIPT because of the large amount of documentation still existing in it, and TeX because it is still widely used in academic areas, where it has still not been supplanted by LaTeX. (LaTeX is to TeX approximately what IBM BookMaster is to SCRIPT.)

nroff and troff are formatters under UNIX**, using a more or less common formatting language; they are briefly introduced for completeness and historical interest.

11.4 The Advantages of Using Document Languages

An unfortunate, almost "religious" dispute seems to have arisen between proponents of document languages and the users of WYSIWYG desk-top publishing programs. The dispute is actually as unnecessary as it is mistaken.

Most desk-top systems save their output in some proprietary formatting language, and the users interact in what has come to be called a "What You See Is What You Get" (WYSIWYG) way, specifying the format and layout of their documentation as they create it.

Traditionally, document language users have created their text and document markup using a system editor, which does not show anything like the finished result; the formatted output could only be seen after the source text and markup were passed through a formatting program.

The proponents of WYSIWYG programs, which tended to be introduced later than document languages, rightly said that the WYSIWYG approach was easier to understand and use, particularly for simple, short documents; definitely for documents where close control of layout is essential, and so on; therefore the WYSIWYG, format-based approach is superior, and should supersede the use of these old-fashioned document languages.

So why are they still used? The answer is in two parts:

- One size doesn't fit all, and

- Actually, WYSIWYG (or something like it) can be used to create document language output.

11.4.1 Why Some Documents Need One Way, and Others Another

There is a very great variety of different types of documents. It is a gross oversimplification, but they can be conceptualized lying on a spectrum of importance of several factors:

- The first is the importance of the appearance of the document compared with the structure of the contents of the document. Both are always important, but the relative importance varies.

  An advertising brochure must be designed very carefully to catch and hold the eye; format and appearance is critical, and must be closely controlled. A format-based approach is almost certainly the best way to produce this.

  That may not be the case with a technical manual; appearance is important, but to the level that it can be controlled by simple rules, and use of appropriate fonts, both of which can be encapsulated in a style definition to a formatter. Here it is more important that information is in the right place — we might be creating an online hypertext version from the same source text as a printed copy; structure is critical, so a document language is more appropriate.

- The number of updates, and the need to keep multiple versions or not, can be an important factor. Simple documents with many versions can be easily handled by format-based systems, but when the documents get more complex, then document languages are probably better; they tend to have specific facilities for version control.

- If the document has to be produced in several versions, perhaps on different output media, or different sizes of paper, this may be easier to do with a document language system. This is particularly true where the document is documenting something that has multiple versions simultaneously — for example, where it describes a product that is produced in more than one type, but the bulk of the documentation refers to all types. A document language allows all versions to be described by a single document, with common text held once, and all variants included by optional text defined as such in the appropriate place.

- Where style may be freely defined by the author, format-based systems probably allow for more freedom over the styles that can be created. Where documents must be consistent with a house style, this is easier to maintain with a document language. Also, if the style has to be changed, with a document language, only the style definition has to be changed; all documents then conform to the new standard - there is no formatting information inside them which has to be changed.

- The life of a document may affect which approach to use. If a document is to be produced once, and never printed again, then it is most appropriate to use whatever is the simplest and easiest way of producing it; that method may, depending on other factors, be a format-based method.

  If the document is to have a long life (measured in years), and might have to be printed again (even though it won't be changed), the *device independence* and *software version independence* become critical. Document languages have these virtues.

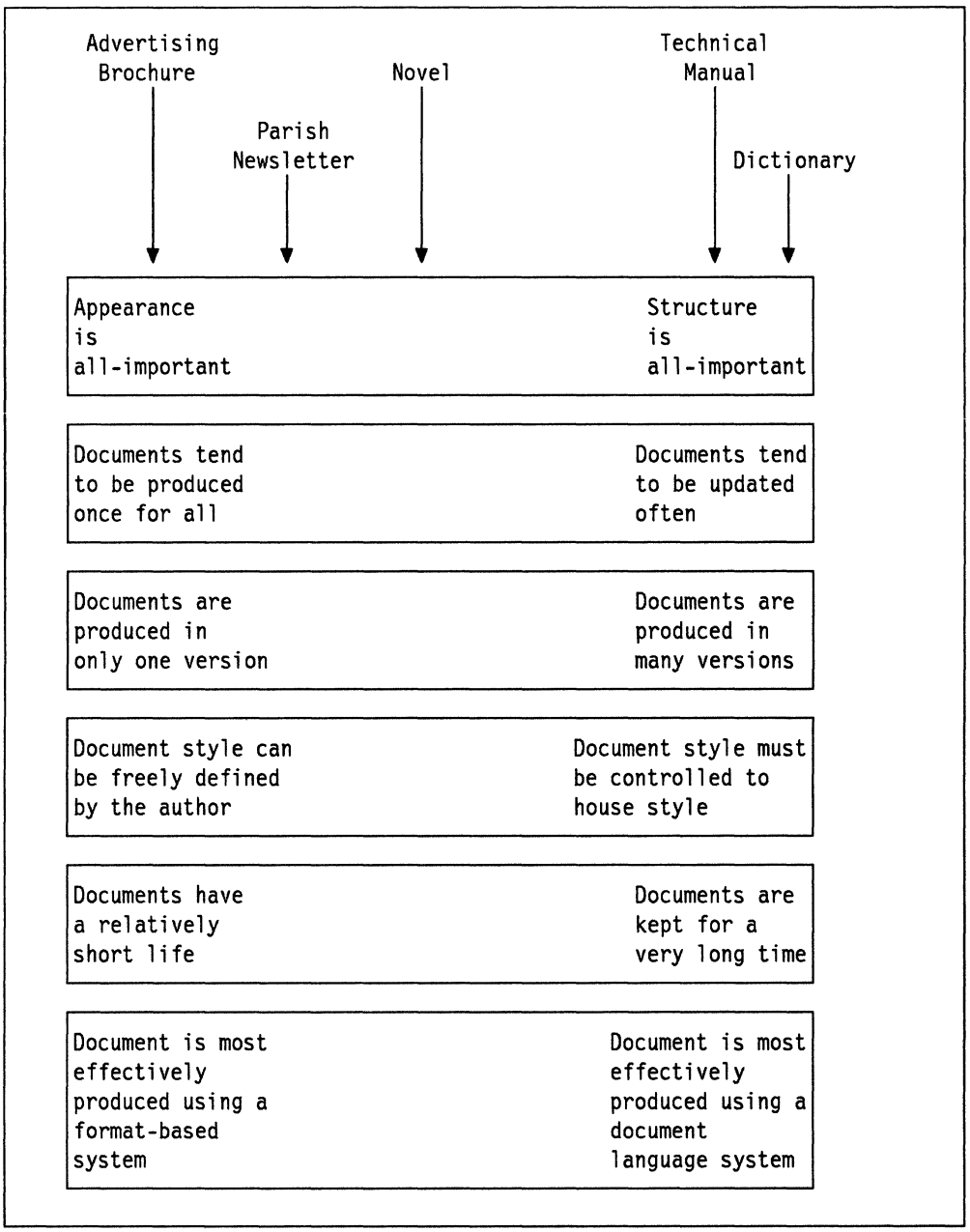


Figure 11. The Document Spectrum

So, it's not always a simple decision, but usually it is reasonably clear that one method is probably a bit better than the other for any particular document. With many, it won't matter too much, which is probably all to the good, as many users will only have one system.

However, a user who is forcing things using an inappropriate system might well find life easier by having both types available.

11.4.2 WYSIWYG Is Always Format-based, Isn't It?

No.

Well, sort of. Since documents produced using a document language only have a specific format when the final form document is actually produced, then WYSIWYG isn't really the term. It should be more of "WYSIWYMG" ("What You See Is What You Might Get.")

However, it is possible to use an editor which shows a possible formatted layout of the document, while what it stores is a document language form, with no format information in there. The editor is like the formatter in this case; it holds a style definition which allows it to interpret the structural information in the document language.

The effect for the user is effectively the same as WYSIWYG, while retaining the operational flexibility of a structure based document language.

IBM sells two such editors which run on the PC, both of which can be used to produce both IBM DCF Starter Set GML and IBM BookMaster:

- IBM MARKUP
- IBM TextWrite

IBM MARKUP is a DOS-based editor, which has a simple, character-based, "structural WYSIWYG" approach.

IBM TextWrite is OS/2 based, and makes much more use of "fancy fonts," color, and other formatting. IBM TextWrite can also be used to produce SGML.

There are also several non-IBM editors which can be customized to produced IBM DCF Starter Set GML or IBM BookMaster.

Nor should it be forgotten that some well designed desk-top publishing programs and word processors can be used to create GML as well as formatted print. It is for example relatively easy to define WordPerfect** macros that will save text as IBM BookMaster. Ventura Publisher** is a desk-top publishing program which saves tagged files; Ami Pro is a word processing program with similar behavior; to convert these to any GML requires an almost trivial filter program.

11.4.2.1 Converting Existing Text

There are also several programs available which will take any formatted ASCII file, and use the formatting context to analyze the structure of the text, and "automatically" tag these files. These programs therefore allow, within limits, any text file to be readily converted to GML; either documents which have been created already, or text created on other systems, or as a by-product of some other process.

One such program is TextTagger. This is a general purpose conversion program which can be used on a wide range of source text, from printer listings to office documents. In its TextTagger/ESA incarnation, it is capable of being customized to a degree that allows of very successful conversion of a wide range of tags on a wide range of source formats.

Its generality can also mean it is not suited for all situations; the customization requires code level skill, so it is probably best suited for organizations that will have to do large quantities of conversions, probably with more than a few input

formats, and that have skilled people available to customize it. In these circumstances, it probably cannot be bettered.

However, where there is only the need for a small quantity of conversion, from only one or two formats, it may be more appropriate to write or have written specific conversion utilities; particularly where the tagging needs are simple.

Also, TextTagger/ESA is not announced in all countries.

Because of these points, we include in A.3, "Conversion from BookManager READ Copy Form to GML" on page 163 a typical example of a conversion program which is capable of adding simple tags to a file, based on the original text format.

11.5 GML Starter Set and IBM BookMaster

IBM BookMaster is not only a product sold to IBM customers, but also the tool used to create the majority of IBM product documentation - the output of one of the world's three largest publishers. This document was created with IBM BookMaster.

Both GML Starter Set and IBM BookMaster evolved originally from SCRIPT, and both use Document Composition Facility as their formatter. It is not surprising that they are fairly similar.

GML Starter Set is, as the name suggests, less rich in its element definitions than IBM BookMaster. IBM BookMaster allows the definition of some three hundred different document elements and components, GML Starter Set perhaps forty or so. The major difference is, however, that IBM BookMaster allows more control over document style without changing the macros used by the formatter.

Both allow from simple to very sophisticated technical documentation to be created. To allow of simple migration, the vast majority of GML Starter Set documents can be processed using IBM BookMaster without change.

```
:h2.Objectives
:p.The primary objective of this document is
to clarify the interrelationships between different print and view
data streams to help make it easier to understand which products will
work together properly.
:p.This can be shown by:
:ul.
:li.Outlining what type of data is contained in each data stream
:li.Outlining how they relate to each other and the I/O devices
that will be used to display or print the data.
:li.Indicating which of the most commonly used programs produce and use
which types of data streams.
:eul.
```

Figure 12. Example of GML Starter Set and IBM BookMaster. This is the markup that created some of the text at the beginning of this book. The :h1. tag defines a header, :p.s define paragraphs, and the :ul. to :eul. defines a list of which the :li.s are individual items (the "ul" stands for "unordered list," as opposed to a numbered one).

11.6 SGML

Generalized Markup Languages, after their initial development by IBM, became increasingly popular for the production of complex, high-quality documentation.

It was also becoming obvious that there should be a standard for interchange of technical documentation between differing types of computer systems, each of which was using perhaps different software, and it was realized that Generalized Markup Languages offered an efficient means of attaining that end.

So the International Standards Organization defined SGML, a "meta-standard" which defines how Generalized Markup Languages should be written. A Generalized Markup Language which can be defined by an SGML Document Type Definition is termed a "SGML conforming GML," and can be understood and processed by any "conforming SGML system."

SGML was adopted by the United States Department of Defense for its *CALS (Computer Aided Logistical Support)* initiative, and the various CALS DTDs are now widely used in the defense industry.

The *IBM SGML Translator* is a program that can transform SGML to GML Starter Set given the SGML Document Type Definition and translate tables. It is delivered with customization for the CALS DTD and an SGML Starter set.

The SGML standard allows the definition of markup languages that define not only the structural elements of documents, but also the ways in which these elements can be combined — in what context specific elements can appear.

It also defines a *Reference Concrete Syntax*, which is a syntax for a markup language in which SGML languages can be written, though deviations from this are allowed in the standard if correctly defined in the DTD and SGML declaration. This flexibility allows existing markup languages such as GML Starter Set and IBM BookMaster to conform to the SGML standard providing an appropriate SGML DTD is declared.

Such SGML DTDs exist, which means that GML Starter Set and IBM BookMaster can be processed on any computer which has programmed for it a conforming SGML formatter.

So, given GML Starter Set, the *IBM SGML Translator*, optionally IBM BookMaster, and appropriate SGML DTDs, an IBM mainframe system can both produce printed and online documentation from SGML source text from any origin (given SGML DTDs and translate tables), and also provide any SGML conforming system with input that it can format to whatever output devices it supports.

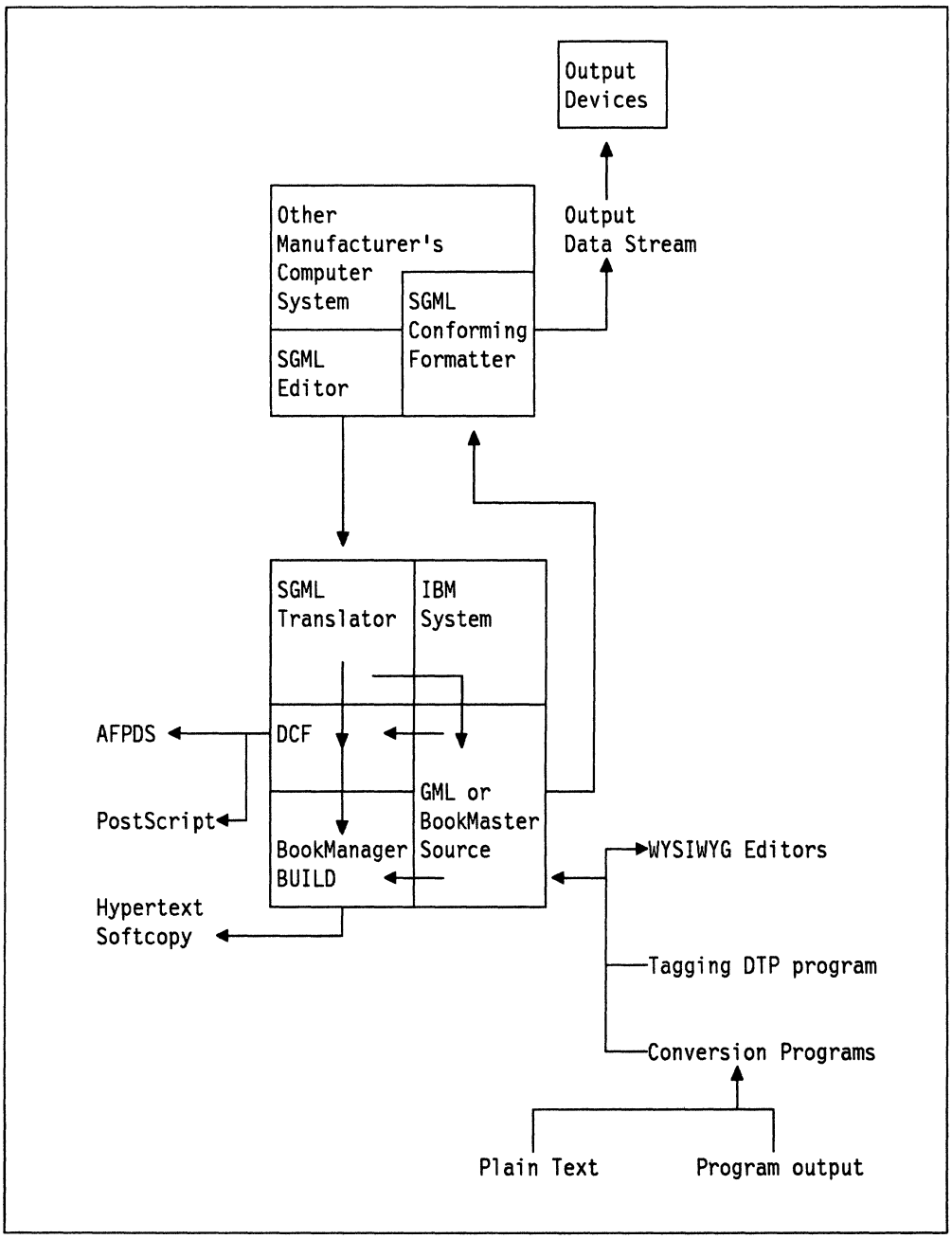


Figure 13. Intersystem Documentation Exchange with SGML

11.7 TEX and LATEX

TEX and LATEX (pronounced "tekh" and "latekh") are products for producing good quality typographic output that originated in the academic and UNIX worlds. They are still widely used in those arenas.

TEX is essentially a formatting language, while LATEX is a macro language based on it, which is more structural and in line with other document languages, though it does at times betray its formatting origins.

The output from a T_EX formatter is in a device independent form called "DVI." This is printed to a particular device with a specific output program for that device, with the name of the output program normally beginning *dvi2*.

These output programs need input bitmap font files at resolutions required for the output device as well as the device independent data stream. This means that to print at any particular point size, the user must have a font bitmap at that size available, or a program to generate such a bitmap.

An exception to the above process is *dvi2ps*, which, if available, transforms the output to PostScript, which is, of course, resolution independent.

There are other programs associated with T_EX, of which the main one is *METAFONT*, used for generating typographic fonts for use with T_EX and LAT_EX output. There are others, for example to create graphics.

As with much of the academic and UNIX world, there is a strong freeware flavor associated with T_EX. Versions of TEX (which from now on will be taken to include LAT_EX) are freely available for both UNIX and PC systems. Commercial offerings also exist on both these platforms, and there is a PRPQ for T_EX on the RS/6000.

Further information, source, and executable code, are available from:

```
TeX Users Group
PO Box 9506
Providence, Rhode Island 02940 USA
```

11.8 XICS

XICS (Xerox Integrated Composition System) is a mixture of document language and formatting language, but is really more formatting oriented than otherwise.

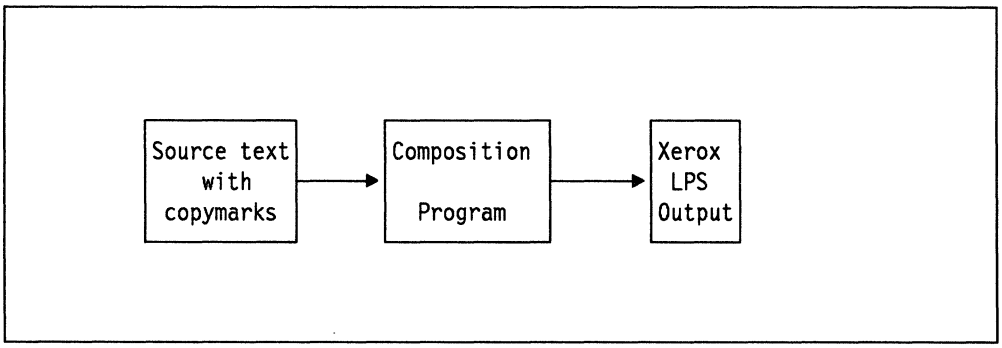


Figure 14. Formatting XICS Source

(Xerox LPS stands for "Xerox Laser Printer System.")

It is processed on Xerox** machines in much the same way as SCRIPT is in an IBM environment. Like SCRIPT, it is a text composition language, and graphic elements are merged in via a macro command.

XICS has a full set of functions for text composition, with boxes and rules, and is capable of generating complex, high-quality typographic output.

General automatic translation of a large number of XICS *copymarks* to SCRIPT commands is possible, though there is not a strict one for one correspondence.

Some manual intervention by a skilled text programmer will almost certainly be needed.

Automatic translation of XICS to SCRIPT or a combination of SCRIPT and GML is possible where a specific set of functions and macros have been employed in XICS; in this case a transform written for this specific subset by a competent text programmer would be capable of translating a larger proportion of the input. However, manual checking and some intervention is again almost certainly essential.

If a conversion to T_EX were required, it is likely that TEX macros could be written to simulate the effect of a set of XICS copymarks and macros used by particular applications. This would not be trivial, however, and would require a competent T_EX programmer.

Translation to pure GML Starter Set, IBM BookMaster, or SGML would be problematic, and should be considered on a case-by-case basis. It would depend very much on what types of copymark and macro were used and how they were used. It is unlikely that precise similarity of output format could be achieved, though functional equivalence should be possible reasonably readily.

For a more detailed introduction to the XICS language, see *Page Printer Migration Programming Guide, S544-3228.*

11.9 nroff and troff

nroff and troff are formatting programs which run on most UNIX systems and IBM AIX*, using essentially the same formatting language.

nroff is used to format text to printers, and troff is used to format to specific phototypesetters. The differences between the two as far as input is concerned are minor ones with regard to machine space units; otherwise the formatting languages are essentially identical.

Both nroff and troff can also be used in conjunction with several preprocessing macro programs, which are used to simplify the use of the formatting language for more complex tasks, like the production of tables or mathematical formulas.

The actual formatting language is very low level, and very similar in nature to the SCRIPT language of DCF.

Some sample formatting controls are:

Control	Meaning
.br	Break output line
.ce n	Center the next n lines
.in +/-n	Adjust indentation
.ls n	Set line spacing
.sp	Space vertical distance
.tr a b	Translate character a to character b
.ul n	Underline next n lines

Preprocessor macro expansion programs are available to make complex tasks simpler in the nroff/troff environment, such as:

Memorandum macros (general document formatting)

Viewgraph macros

Table macros

Mathematics equation macros

There are so far as we are aware no transforms currently available to convert nroff/troff input to any other format; if this were required, a change to SCRIPT controls should be relatively simple to define.

Chapter 12. Print and View within the Different Environments

This section contains an overview of the different platforms where Advanced Function Printing is done, and a description of how PSF handles different object content architectures to achieve optimized performance and minimize the work load on the processor during printing.

AFP printing can be done with one of the following printer drivers:

- MVS PSF
- VM PSF
- VSE PSF
- AS/400 Print Services
- PSF/2
- IBMAFP AFPDS Drivers for OS/2 and DOS/Windows
- RPM V.2.0
- RPM V.3.0

PSF optimizes print performance with these content architectures:

- IOCA
- GOCA
- FOCA for 4028

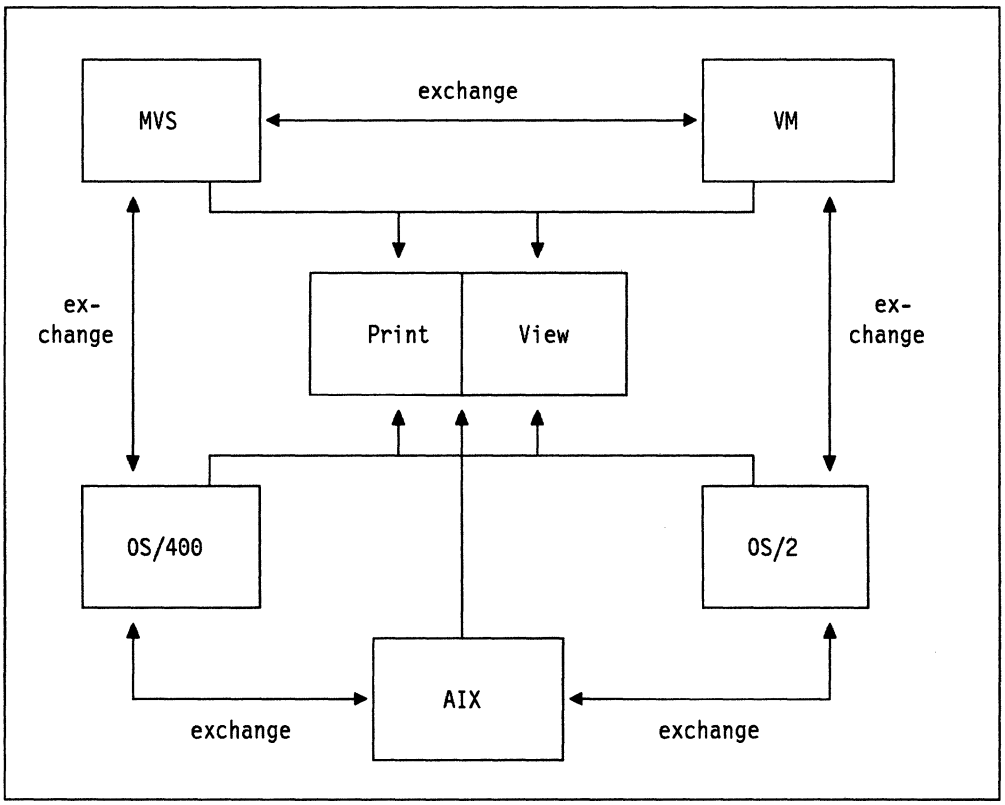


Figure 15. SAA Model for Print and View. An enterprise-wide solution for printing and viewing demands a solution to work on all platforms available under SAA. Users want to exchange data to print and view between platforms, and they expect no additional conversion activities. Working within the rules of SAA provides this solution and it is extendable to other platforms as they become available under SAA.

12.1 Basic PSF Functions

Print Services Facility is the printer driver for AFP printers. These printers include:

- IBM 3900
- IBM 3835
- IBM 3828
- IBM 3827
- IBM 3825
- IBM 3820
- IBM 3812
- IBM 3816
- IBM 4028
- IBM 4224
- IBM 4234
- IBM 3800-3

The functions PSF provides, amongst others, including:

- Interfacing to the spool
- Combining and merging
  - Print data
  - Fonts
  - Images
  - Overlays
- Two-way dialog between printer and the driver
- Error recovery
- Different attachments for the printers

PSF includes all the necessary resources to switch from a line printer to an AFP printer without the need for any program change. The printed layout is the same as on a line printer but with the improved quality possible with an AFP printer.

This is accomplished by providing a FORMDEF, which contains all the information for a physical page description, and a PAGEDEF, which contains the information on how and where to position data from an application program onto the page.

The base package also includes compatibility font objects.

PSF provides the means to move formatting out of a program.

12.2 How PSF handles Object Content Architectures

When a print job is submitted to PSF it first establishes a dialog with the printer that the user has directed the output to.

PSF must have answers from the target printer to questions like:

- Who are you?
- What are your capabilities?
- How do you expect IPDS?
- What resources are available in the printer?

A printer gives answers to PSF to those questions and PSF uses them to:

- Optimize the printing process
- Make sure that the printed page has the same appearance on any AFP printer, wherever it is physically located.
- Decide how to print when the printer can't handle a function activated by a definition.

  For example, if duplex printing is requested on a fanfold printer, PSF instructs the printer to print the back side page on the next sheet. The user is guaranteed that the printed page has exactly the same layout as it would if duplex were possible.
- Reduce workload on the processor, if the printer is able to do some work by itself.
- Where possible during printing, and where the printer is capable, to download resources only once and keep them ready for printing inside the printer.
- Only download resources when necessary. If they are already available or in the printer PSF will not download them.

  For example, the IBM 4028 printer has over 30 fonts at a resolution of 300 pels available in the printer. If these fonts can be used, PSF just sends the control information on which font to use.
- The printer tells PSF that it has the features installed to decompress IOCA in the printer or to accept GOCA, PSF will download IOCA in a compressed form, or will download data in GOCA format.
- Handle error recovery. PSF knows at every moment where physically a form is on the paper path. If there is a paper jam, PSF works together with the printer and the controlling subsystem (JES, for example) to set up the printing process again, starting with the page that was the first one not to be placed in the output bin.

  The error recovery is very powerful. No operator intervention is needed to restart a job on the right page, or to even switch printing to another printer which is also driven by the same instance of PSF, and to continue printing starting at a checkpoint (CKPT).

12.3 Advanced Function Printing (AFP) on Mainframes (MVS, VM, VSE)

The basic concept of Advanced Function Printing on a mainframe is the same in the different operating systems.

1. PSF as printer driver accepts as input:

   1403DS
   AFPDS
   1403DS and AFPDS mixed

2. PSF is provided via job control with the necessary information: what data, with what resources, and where on a page to print.

   FORMDEF (physical page description)
   PAGEDEF (logical page description)
   Overlays, referenced in FORMDEF or documents
   Page segments, referenced in overlays or documents
   Fonts, referenced in PAGEDEFs, in overlays, or in documents

3. PSF reads the input data from the spool and converts it into the IPDS data stream.
4. IPDS is used for communication between PSF and the printer and for transporting all data and resources to the printer.
5. The printer is able to interpret IPDS:

Where PSF works:

- Within MVS, PSF is a functional subsystem under JES.
- Within VM as virtual machines:
  - SFCM (Spool File Control Manager)
    There is one of these virtual machines in a system. It checks the input, data, and resources and passes them to the appropriate printer driver according to the printer choice specified by the user.
  - PDMxx (Printer Driver Machine)
    There is one of these virtual machines for every AFP printer on the system. It receives its input from the SFCM1 machine and takes care of the actual printing. An "xx" in the name means a number internally assigned to a printer an defined by the system programmer.
- Within VSE, PSF runs in a partition

For printing on printers that are attached via communication lines VTAM* is required.

12.3.1 Mainframe Model

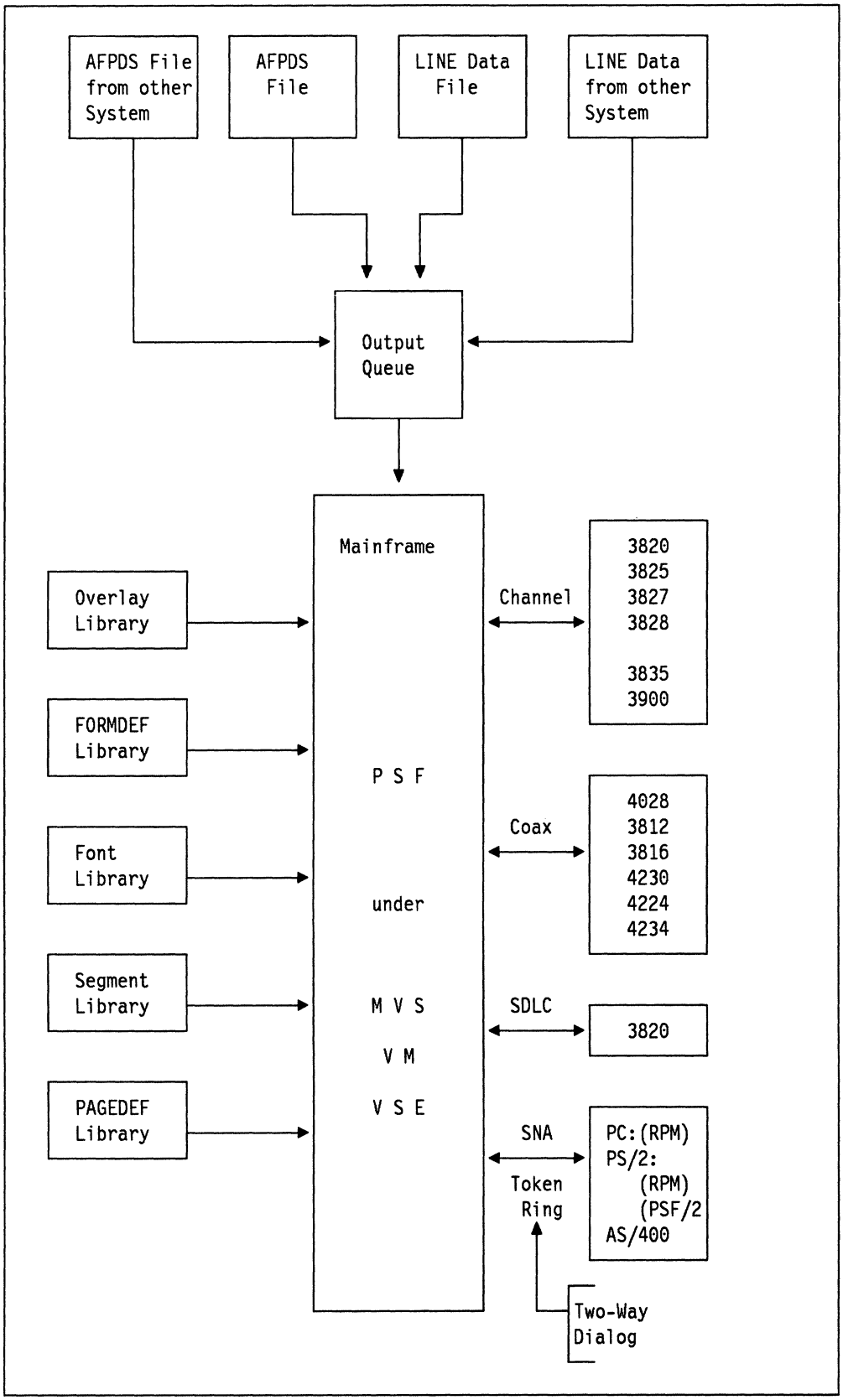


Figure 16. Mainframe Model for AFP Printing

12.4 IBM Print Services Facility Version 2

Version 2 adds to the functions so far described.

The IBM LaserPrinter 4028 Model NS1 is supported and can print using downloaded 300 pel resolution host fonts from System/370* or by accessing resident fonts in the printer. Font processing is aided by the use of a new batch utility and font metrics provided with PSF. The font metrics assist in formatting documents when using resident fonts. The batch utility is used to convert existing 240 pel font libraries to 300 pel. This conversion aids users in printing on the 4028 using PSF.

Full PSF support is provided for the IBM 3900 Advanced Function Printer, equivalent to the IBM 3835.

PSF has been functionally enhanced to enable or facilitate new business application solutions:

- Support for printer-based vector graphics, image decompression, image rotation, image clipping, and image scaling to allow reduction/enlargement. Printer-based bar codes are made available with optional human readable characters. Transformation capability from IM1 image to IOCA and uncompressed IOCA to IM1 provides additional flexibility and interchange potential.
- Ability to position electronic overlays dynamically on each page of output. In previous releases, overlays were limited to the same fixed origin point on every page.
- Enhanced duplex printing support. Users can now specify different horizontal and vertical offset values for the front and back sides of pages. This facilitates printing on punched paper, for example.
- It is now possible to specify printing of an overlay on the front or back side of a sheet with no variable data. In previous releases, an application change was required to insert a blank record for each side with no variable data.
- Using these new functions in the AFP resources is supported in a new release of the Page Printer Formatting Aid program product.
- The Core Interchange fonts include typographic and uniformly spaced fonts which provide a rich selection of point sizes, typefaces, and national language character sets for a variety of print applications.
- Support for color on the 4224 and a quality selection function on the 4224 and 4234.
- Envelope selection for the IBM 4028.
- All PSF Version 2 print driver functions are available in the base package. No additional features need to be installed.

12.5 Font Pruning

Performance considerations

The number of characters in the Core Interchange and Proprinter Emulation character sets is greater than the number of characters in current IBM-supplied Compatibility Font character sets. Therefore, use of *font pruning* may be required for complex print applications using many fonts.

The *font pruning* capability provided with Version 2 of PSF/VM and PSF/MVS downloads only the characters required for the requested

code page. This reduces the amount of data sent to a printer when loading fonts.

In cases where the additional CPU time required to analyze the fonts outweighs the print savings, *font pruning* can be turned off. In either case, overall printer resource management is improved and greater efficiency realized.

12.6 Operating System/400 Advanced Function Printing

OS/400 includes support for AFP printers as part of the basic operating system. The following sections describe OS/400 AFP implementation and AFP functions available to the user in an OS/400 environment.

12.6.1 Support for Medium to High Speed Page Printers

The OS/400* licensed program now includes PSF-like support for IBM 3812-002, 3816-01S, 3820, 3825, 3827 and 3835 Page Printers attached to an AS/400 system.

OS/400 software support for the IBM 3820, 3825, 3827 and 3835 Page Printers uses the same interfaces currently used for the IBM 3812 and 3816 Page Printers, including selection of fonts, page rotation, computer output reduction, drawer and code page.

An AS/400 system can now be used as a remote print server for System/370 generated Advanced Function Printing Data Stream (AFPDS). It is the customer's responsibility to transfer AFPDS from a System/370 to an AS/400 system with file transfer facilities.

The CL command (PRTAFPDTA) is provided to move this file onto an OS/400 output queue. System/370 generated overlays, page segments, and form definitions can be downloaded from a System/370 and stored on an AS/400 system for later use.

The AS/400 licensed program, IBM AS/400 Advanced Function Printing Fonts (5728-FNT), offers System/370 equivalent families of fonts for printing AFPDS on an AS/400 system.

OS/400 commands DSPSPLF and CPYSPLF are not supported for AFPDS. On an AS/400 system, AFPDS can be printed only on the IBM 4028, 3812-002, 3816-01S, 3820, 3825, 3827 and 3835 Page Printers.

Printing of System/370 generated 1403 line data on AS/400 attached printers is supported with RJE. Use of System/370 page and form definitions is not supported with 1403 line data on the AS/400 system.

Printing of System/370 generated AFPDS graphics, image and bar codes is supported on AS/400 attached IBM 4028, 3812-002, 3816-01S, 3820, 3825, 3827 and 3835 Page Printers. Printing of AS/400 generated graphics, image and bar codes is supported on AS/400 attached IBM 4028, 3812-002 and 3816-01S Page Printers, and is not supported on AS/400 attached IBM 3820, 3825, 3827 and 3835 Page Printers. If required, solutions from AFP partners are available.

The IBM 3820, 3825, 3827, and 3835 Page Printers will attach to the AS/400 System Units through the IBM Token-Ring Network with the Remote PrintManager* Version 2.0.3 support. The IBM 3820 Page Printer will also attach

to the AS/400 through an SDLC communications line. The IBM 3820, 3825, 3827, and 3835 Page Printers attach to all models of the AS/400.

12.6.2 How to Start

For a user to work on AS/400 with AFP, it is necessary to establish the correct environment:

1. Check for a correct Library List
   Library **QAFP** and available **font libraries** are required.
2. The command: **GO AFPU** calls the menu, which allows you to start working with the AFP utilities.

12.7 AS/400 AFP Model

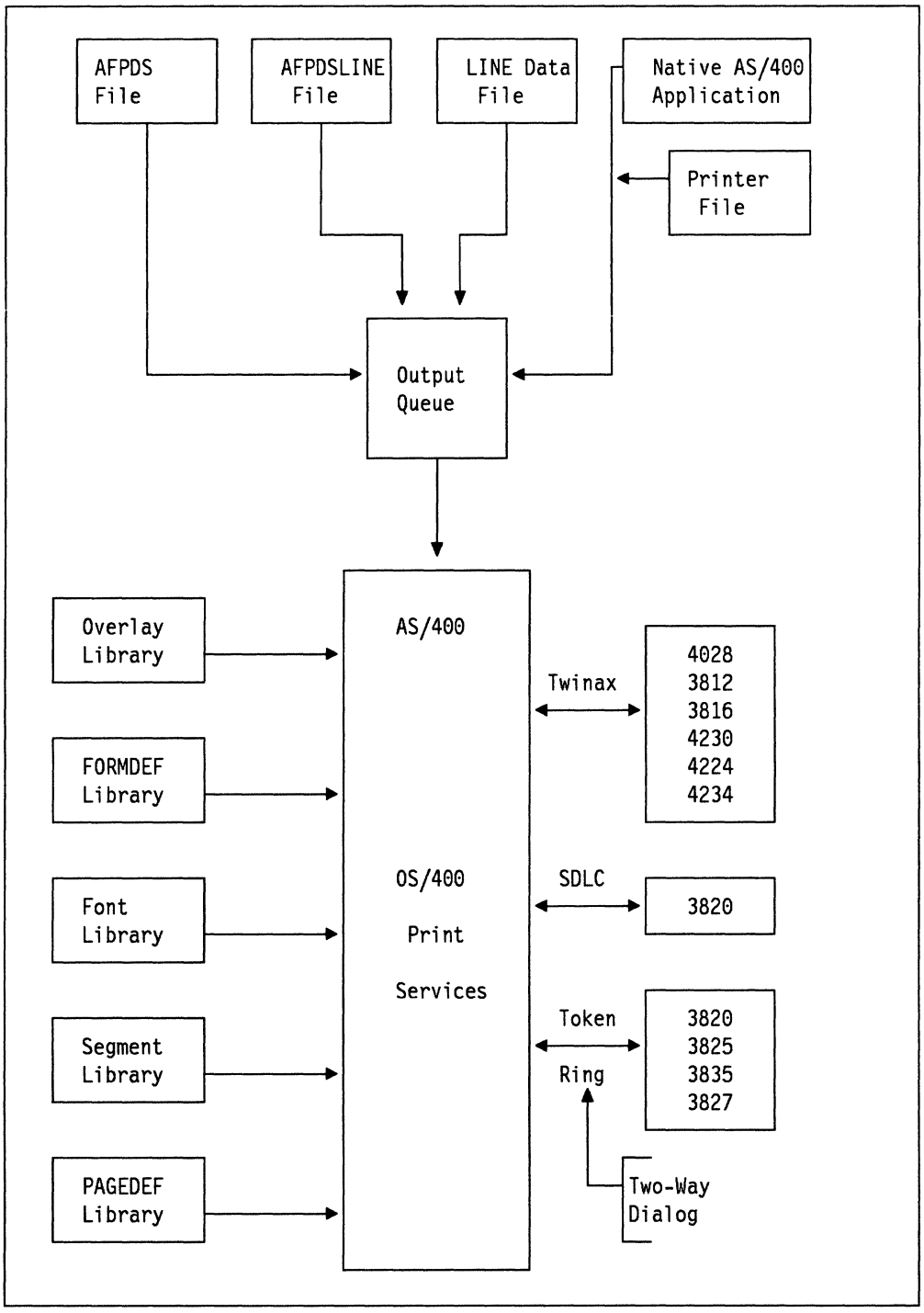


Figure 17. AS/400 Model for AFP Printing

12.8 Print Output in AS/400 Systems

Four different types of output in an AS/400 System can be directed to an AFP printer. Each output type uses a different interface, to reach the printer Although a description of data streams can be found elsewhere in this book, here is a description from the viewpoint of an AS/400 user.

SNA Character String (SCS) data stream The SCS print data stream is based on a set of single- and multiple-control codes inserted with the print data. OS/400 creates the SCS data stream based on the request of the application program and places the SCS print file on the output queue.

If this output is destined to be printed on an SCS printer, the SCS print file is sent to an SCS printer. However, if the SCS print file is to be printed on an AFP printer, OS/400 converts the SCS data stream to Intelligent Printer Data Stream (IPDS).

This conversion lets existing AS/400 applications be printed on AFP printers without change.

The new support in Version 2 Release 2 of DEVTYPE(*AFPDS) generates native AFPDS, eliminating the SCS data stream conversion.

AFPDS The AFPDS data stream provides an application interface for AFP applications. This fully paginated data stream is device independent and enables the construction of pages composed of text, image, and graphics objects.

IPDS The IPDS data stream is generated by the OS/400 Print Services and is similar to the AFPDS data stream; however, the IPDS data stream contains a dialog component to communicate with the printer in a two way conversation.

Compared to the AFPDS data stream, the IPDS data stream is bound to a specific printer device, which means it is generated by the printer driver (OS/400) and must be bound to the device that will print it.

Line data output The line data stream dates back to the 1403 printer and is commonly used for printing on line printers. The data stream is composed of a single-byte character in the first byte of the print record followed by the data to be printed.

The control characters instruct the printer to do basic functions, such as skipping to a specific line on the page or skipping to a new page.

AFPDSLINE data stream This data stream is a hybrid of line data and AFPDS data mixed into the same data stream. This data stream can be interpreted by PSF.

This type of data is often created when an existing LINE data application is enhanced by inserting an occasional AFPDS structured field record in the line data output to perform function such as printing an overlay. **The printer driver for AFP Printers accepting IPDS data streams is part of the operating system within AS/400.**

12.9 Advanced Function Printing Utilities/400

The IBM Advanced Function Printing Utilities/400 provides interactive, menu-driven facilities that allow Application System/400* users to design, create, and manage advanced function printing (AFP) resources such as overlays (electronic forms and labels) and page segments (logos and images) natively on the AS/400 system for the users of intelligent printer data stream (IPDS) printers.

The AFP Utilities creates AFP resources that conform to the Advanced Function Printing Data Stream (AFPDS) on the AS/400 system. It also provides the facility to print users' data from a database file in various formats with various fonts and bar codes on the IPDS printer without developing any application programs. As an example, it allows the customer to print bar code labels from data stored in the database file.

With this program, a non-programmer user can design, create, update, and manage a variety of forms and labels, and print users' data using those forms and labels.

The AFP Utilities provides the support needed to take full advantage of All-Points-Addressable (APA) IPDS printer capability for AS/400 users. Its functions is equivalent to the System/370 Advanced Function Printing (AFP).

Through the near WYSIWYG (What You See Is What You Get) editor, the user can interactively design, create, and verify AFP resources such as overlays.

With the AFP Utilities and a PC image program such as:

- Image Data Utility (PC/IDU) (5785-EDW)
  with IBM Image Support Facility 2 (5669-197)
- IBM ImagEdit V2.0 (75X3255)
- OS/2 Image Support (49F4608)

users can easily design logos and images on the PC, and store them in the folder as PC documents through the IBM PC Support/400 shared folder function. Then users can convert the image to a page segment and include it in the overlay. Users can also scan an existing preprinted form on the PC, print the scanned image with the grid, and then by referring to the printout, use rs can easily design and create the overlay from it.

The AFP Utilities also provides the capability to print users' data in a database file in various formats with various fonts on IPDS printers without developing any application programs.

The IBM Advanced Function Printing Utilities/400 V2 is comprised of an integrated set of advanced print functions:

1. Overlay Utility
2. Resource Management Utility
3. Print Format Utility

Overlay Utility provides an interactive overlay resource design and creation function. It allows users to design an overlay by the near WYSIWYG editor that provides the approximate image of the printout on the screen and store it as a source overlay.

The source overlay is created, and stored as a member of a physical file. Users can modify the source overlay by the editor on the design overlay screen which shows the approximate image of the form, and build the overlay resource from it.

The overlay resource created by Overlay Utility conforms to the Advanced Function Printing Data Stream (AFPDS).

Resource Management Utility (RMU) provides the integrated AFP resource management functions for overlays and page segments.

Users can create a page segment from either a PC document in a folder or a database file member. With the PC support shared folder function, users can store PC files in the folder of the AS/400 system. Users can use folders from the PC as if they are PC disks directly connected to the PC. In this way, users can use a PC image product such as Personal Computer/Image Document Utility (PC/IDU), ImagEdit, and OS/2 Image Support to scan logos and preprinted forms, edit the scanned images, and store them in the folder as PC documents. With the ES/9370*, System/370, or System/390* connected to the AS/400 system through communication lines, users can use the product such as GDDM and Image Handling Facility (IHF) to create an image file, send it to the AS/400 system, and store it in the database file member.

Then, by using the page segment creation function of the RMU, users can convert the image data in the page segment format so that it can be included in the overlay.

Note: The image data stored in the PC document or database file member must be in Image Object Content Architecture (IOCA) format (Image Data Stream (IMDS)).

RMU also allows users to copy, delete, print and display the description, and change the text of AFP resources from the list of AFP resources found inthe selected libraries.

Print Format Utility (PFU) provides the function to print a member of a database file in various formats with various fonts and bar codes on IPDS printers without developing any application programs.

Users can design a record layout and page layout using the near WYSIWYG editor similar to the overlay editor, and store it as a Printout Format Definition (PFD) definition. Users can update the PFD definition with the editor to change the printout format.

When printing the database file member, users specify the PFD definition name so that its corresponding print format is used. PFU provides the function to perform record selection so that the printout contains only selected records.

12.10 Advanced Function Printing Fonts/400

Document processing and publishing applications require a large variety of proportional typefaces in order to satisfy demands for aesthetics, variety of style, emphasis, and readability.

The Advanced Function Printing Fonts/400 Version 2 contain fifteen separately orderable font features.

- Sonoran Serif
- Sonoran Serif Headliner
- Sonoran Sans Serif
- Sonoran Sans Serif Headliner
- Sonoran Sans Serif Condensed
- Sonoran Sans Serif Expanded
- Monotype Garamond**
- Century Schoolbook**
- Pi and Specials
- ITC Souvenir**
- ITC Avant Garde Gothic**
- Mathematics and Science
- DATA1
- APL2
- Optical Character Recognition

12.11 IBM Database Publisher/DOS for the AS/400 Version 2

Database Publisher integrates the IBM AS/400 native data base with desk-top publishing programs. The process is automated and data-driven and enables publication of documents as business needs dictate.

Once AS/400 system data needed to publish is defined, and the document format is chosen, Database Publisher can be run in "batch" mode as an extension to an existing AS/400 system or DOS application.

Using IBM Interleaf** Publisher, Aldus** PageMaker** or Ventura Publisher, the user defines how the final document should look. After identifying the data needed from the AS/400 data base and sending it to the workstation through PC Support, a decision is made on how to convert the data using Database Publisher. The instructions specified are converted to a "recipe." The recipe can be called as a DOS process and can build a publication or partial components automatically on demand.

12.12 Advanced Function Printing (AFP) on Workstations

Advanced Function Printing was not originally available on workstations. There was a need to provide it for two main reasons:

1. To print remotely with high quality and on high volume printers
2. To integrate workstation users into high quality and high volume printing.

Development of workstation printing was went with the following products:

Remote Print Print Manager Version 2.0
is a program running under DOS. A PS/2* or a PC-AT* with a /370 channel card is required. RPM V.2 establishes a fixed connection to PSF on the mainframe and the printer, though PSF is the actual printer driver.

To improve performance printing with RPM V2.0, all of the required resources such as fonts, overlays, and page segments can be stored on the hard disk of the workstation at any time, and they are kept there for permanent use.

During printing then, no down loading of resources decreases performance and blocks other users sharing the same line.

Remote Print Print Manger Version 3.0
is a program running under DOS. A PS/2 with a /370 channel card is required. RPM V.3 is divided into several programs to accomplish host data printing as well as printing work station data.

There are three main programs. They can work one at a time in one, two, or three workstations in parallel depending on the workload. All programs is given access to the same hard disk for reading or writing, depending on the task.

1. **Host Print File Receiver**
   PSF establishes a dialog between the printer and itself. This program tells PSF on the mainframe to be the printer. PSF spools print data to a program and this program stores the IPDS data stream on hard disk, thus providing remote spooling.
2. **LAN Print File Receiver Converter**
   This program receives ASCII data from LAN users. Then they are converted and send to the hard disk, where the printer server has access to it.
3. **Printer Server**
   This program has access to the print files sent down from the mainframe and to converted ASCII data from the LAN and can send both to the printer.
   It is also able to mix an overlay from the host with converted ASCII data and then print it.

PSF/2 Version 1.0
is a program providing full AFP functions on a workstation integrated into a LAN. It runs under OS/2. Its functions are comparable to those of PSF. It supports the following data streams as input:

- ASCII
- AFPDS
- Metafile

PSF/2 supports many different output devices. So it has to provide different output data streams depending on the printer it has to support:

- Channel-attached printers
  - IBM 3900
  - IBM 3827
  - IBM 3825
  - IBM 3820
- Coax-attached printers
  - IBM 4028
  - IBM 3812
  - IBM 3816
- LPT1
  - IBM 4019
  - IBM 4029
  - Printers supporting:

- PPDS
- HP-PCL4

Using channel-attached or coax-attached AFP printers only the two way dialog between printer driver and printer is available. Printers attached via LPT1 receive a complete page in the form of a bit map which is automatically prepared by PSF/2.

PSF/2 Version 1.1 This version provides the full set of functions of PSF/2 V.1.0. The main differences between PSF/2 V1.0 and PSF/2 V.1.1 are:

- PostScript is accepted as an input data stream
- HP-PCL5 is supported as an output data stream
- Printing from the main frame in a LAN is done automatically.

AFP Viewer This program runs under Windows** and allows you to display files as an alternative to printing them. Files that can be displayed include AFPDS files, page segments, overlays, as well as ASCII files. While a file is displayed you can:

- Clip a portion of the displayed page and scale the clipped area.
- Copy one or more pages from an AFPDS document into a new AFPDS document.
- Convert a page or clipped area to an AFPDS overlay.
- Print one or more selected pages or the clipped area.
- Change which AFPDS form definition is used to display the file.

The AFP Viewer can only view documents that are stored on the workstation's disks. Host files must be downloaded before they can be viewed. The AFP viewer itself does not provide any facilities for downloading files.

12.13 Remote PrintManager Version 2.0

Remote PrintManager is an IBM Personal Computer-based program that allows print resources such as fonts, overlays, and page segments to be made resident at the location of a remote printer.

Version 2 of Remote PrintManager extends support to the IBM 3820, 3825, 3827, and 3835 Page Printers and gives the user a choice of remotely attaching the printers via either an SDLC communication line or token-ring attachment using SNA LU 6.2.

Remote PrintManager Version 2.0 allows the printing of double-byte character sets on those RPM supported printers that provide double-byte support.

Remote PrintManager performs three main functions:

1. Pass-through emulation. Remote PrintManager receives records from PSF, transmits the records unchanged to the printer, and receives printer responses and error signals and transmits them back to PSF.
2. Remote library management. This function records and informs PSF of resources available at the remote site and downloads them to the attached printer.

3. Resource object collection. This function scans the data stream targeted for the printer by PSF and copies resources marked "public" by the user to the Remote PrintManager V2 Resource Library.

12.13.1 Remote Print Manager Version 2.0

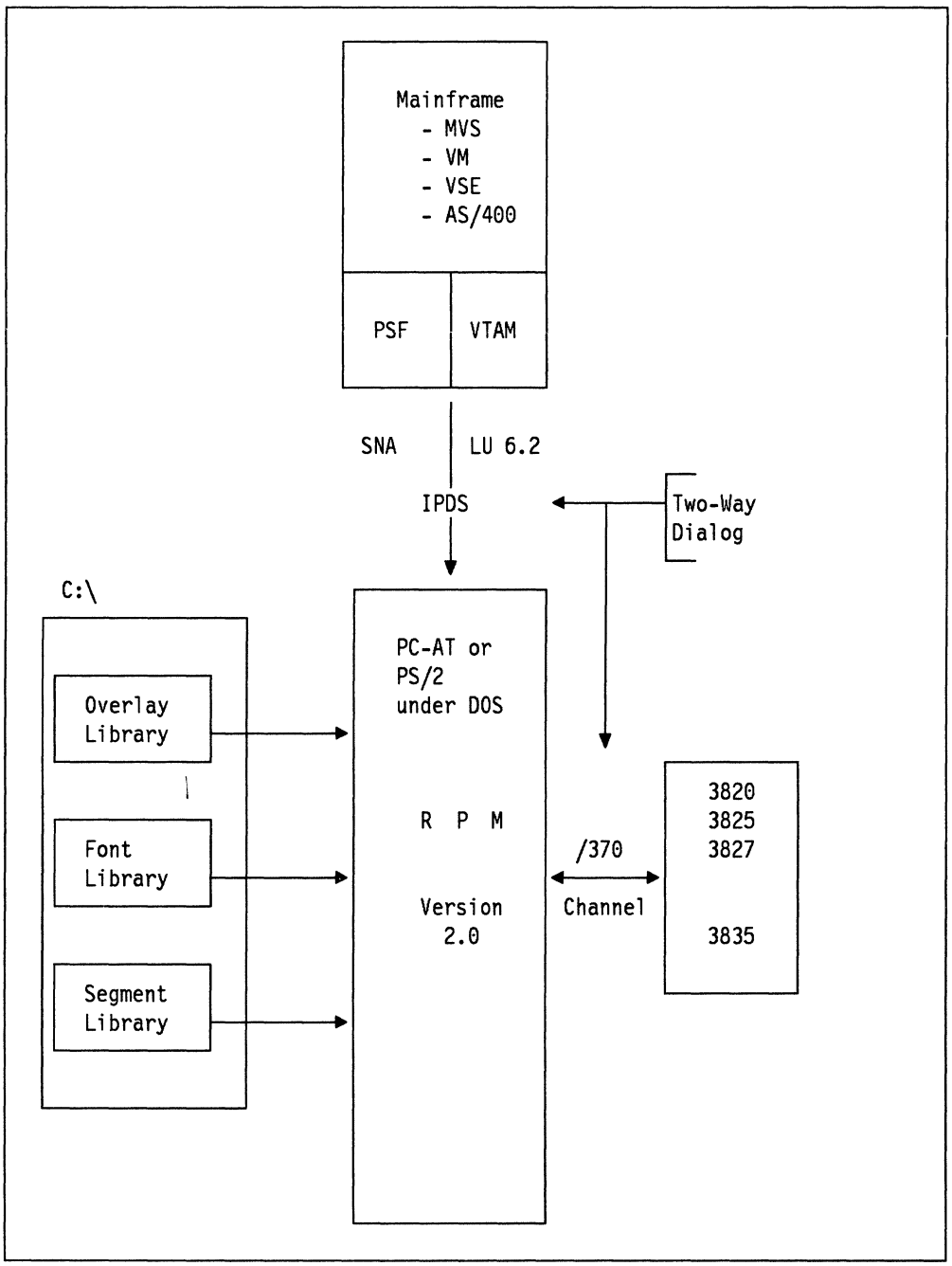


Figure 18. RPM V.2 Model for AFP Printing

12.14 Remote PrintManager (RPM) V3.0

Remote PrintManager V3.0 allows the remote connection of the IBM 3820, 3825, 3827, and 3835 page printers. RPM V3.0 uses an IBM Personal System/2 equipped with a System/370 channel emulator card and a communications card. It can:

- Manage a remote print spool from the print server workstation

- Download jobs from the host at night and place them on the spool for printing the next day
- Attach to PS/2 machines which use the Micro Channel* bus
- Share the printer between the host and other workstations on a LAN
- Merge ASCII data with forms overlays downloaded from the host

12.14.1 Remote Print Manager Version 3.0

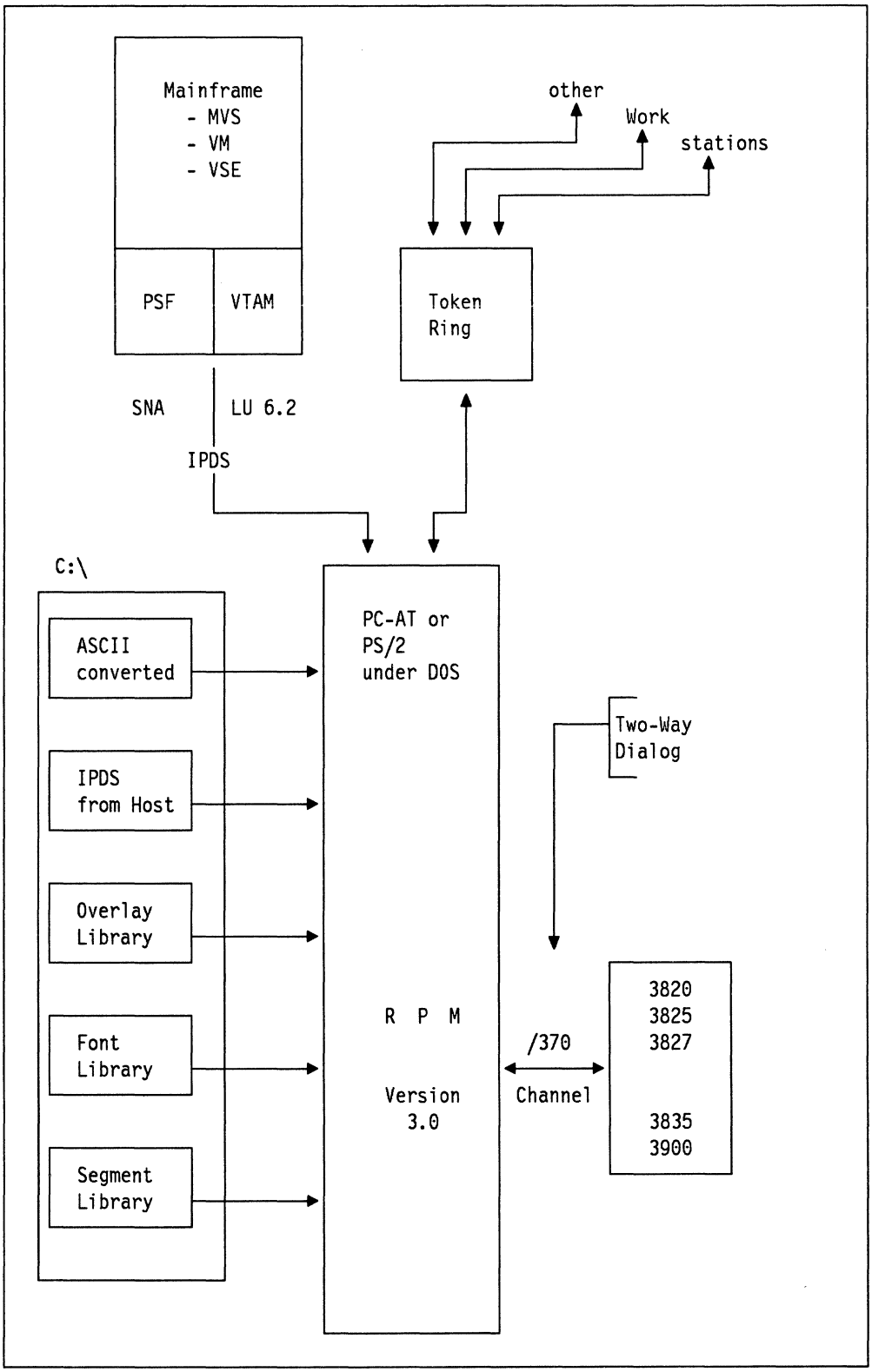


Figure 19. RPM V.3 Model for AFP Printing

12.15 Print Services Facility/2

Print Services Facility/2 (PSF/2*) Version 1.0 functions as a LAN print server, supporting multiple workstations, data streams, and printer types.

Version 1.1 allows both host and workstation application data to be printed on LAN-attached printers. It also provides a transform that processes Adobe Type 1 fonts.

12.15.1 VERSION 1.0

PSF/2 Version 1.0 serves as either a printer driver on a stand-alone system or a print server for a local area network (LAN). It may also be used to print host sourced AFP files if the print jobs and resources are manually downloaded. It supplies the following functions:

- Support for the following input data streams and applications:
  - Support for Microsoft Windows** 3.0 applications
  - Support for OS/2 Presentation Manager applications
  - ASCII (Proprinter* II and QuietWriter III emulation)
  - Advanced Function Printing Data Stream (AFPDS)
  - Mixed Object Document Content Architecture Presentation Interchange Set 1 (MO:DCA-P IS/1) with the following data objects:
    - Bar Code (BCOCA)
    - Graphics (GOCA)
    - Image (IOCA)
    - Presentation text (PTOCA)
  - OS/2 Graphics data in metafile format
- Resource management and error recovery
- A rich selection of AFP fonts:
  - IBM Core Interchange Fonts (240 and 300 pel)
  - IBM-supplied compatibility fonts (240 and 300 pel)

PSF/2 supports the following AFP printers:

- IBM 3812 PagePrinter Model 2
- IBM 3816 PagePrinter Models 01S and 01D
- IBM LaserPrinter 4028 Model NS1
- IBM 3820 Page Printer
- IBM 3825 Page Printer
- IBM 3827 Page Printer
- IBM 3835 Page Printer
- IBM 3900 Advanced Function Printer

PSF/2 also supports the following non-AFP printers:

- IBM LaserPrinter 4019 and 4029 (supported as a 4019) in either:
  - IBM Personal Printer Data Stream (PPDS) mode
  - Hewlett-Packard Printer Command Language (HP PCL4**) LaserJet** emulation mode
- Hewlett-Packard LaserJet printers and other compatible printers that accept the HP PCL4 data streams.

  Note: PSF/2 generates HP PCL4 and therefore supports HP PCL5 printers as HP PCL4 printers.

12.15.2 Print Service Facility/2 V.1.0

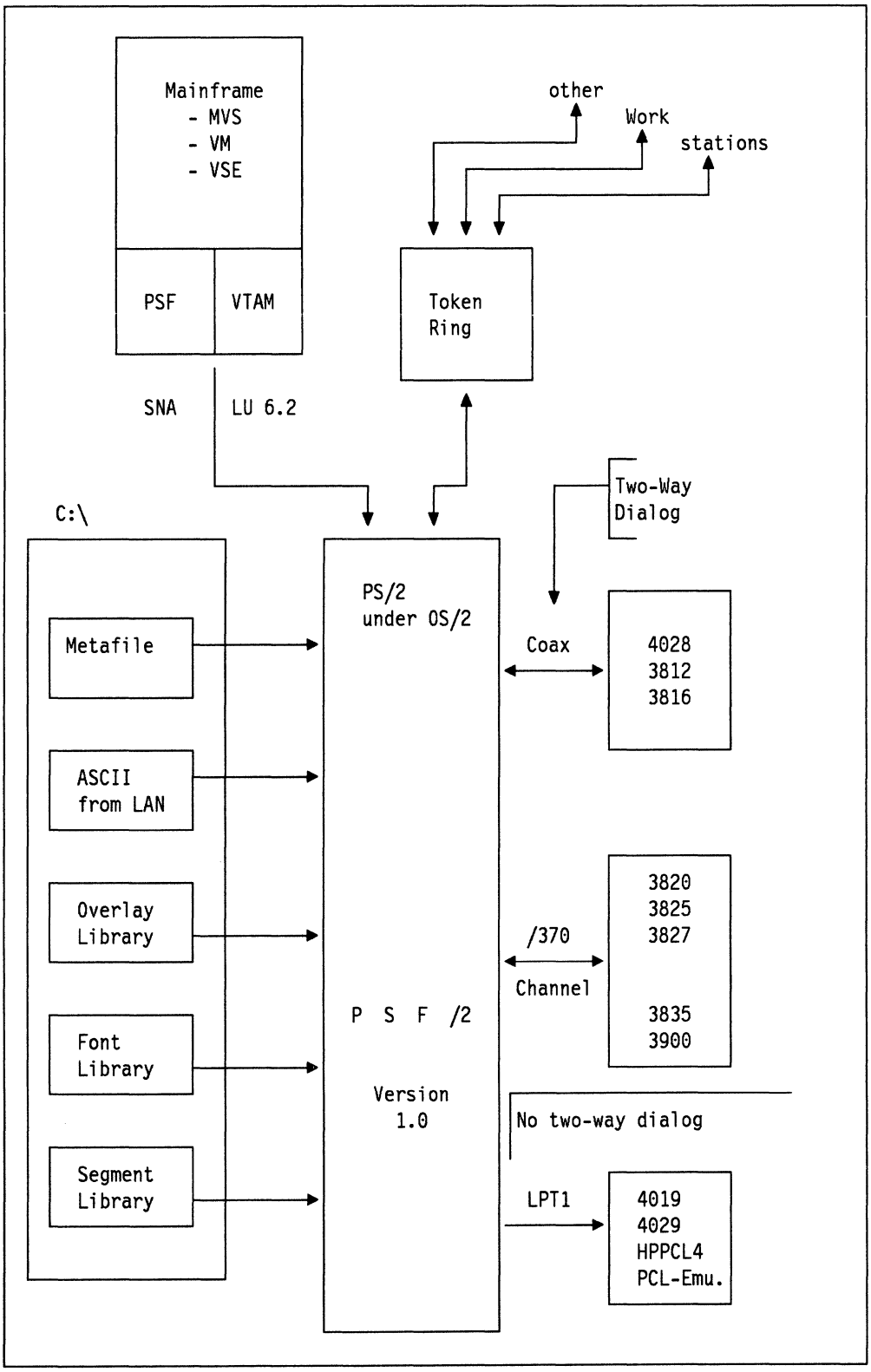


Figure 20. PSF/2 V.1.0 Model for AFP Printing

ASCII, AFPDS, and metafile data stream support means that DOS, UNIX, and OS/2 applications may be accommodated, and can take advantage of AFP

functions such as merging forms overlays and page segments (logos, graphics, signatures). IBM Print Services Facility/2 also supports printing of MO:DCA-P IS/1 documents generated by ImagePlus applications.

PSF/2 provides an OS/2 Presentation Manager device driver that generates AFPDS for applications that use the OS/2 PM Graphic Programming Interface. The AFPDS can then be printed by PSF/2.

PSF/2 provides a Microsoft Windows 3.0 device driver that generates AFPDS for applications that use the Windows 3.0 Graphic Device Interface. The AFPDS can then be directed to the LAN for printing by PSF/2. This driver enables Windows 3.0 users to use IPDS printers as well as the error recovery and job management capabilities of PSF/2.

In addition, PSF/2 can be used in conjunction with OS/2 Transmission Control Protocol/Internet Protocol (TCP/IP) to support printing from UNIX clients, including IBM Advanced Interactive Executive (AIX*).

Printer sharing between PSF/2 and the OS/2 Print Manager is also possible for the 4019, 4029, and HP PCL4 printers. This enables current OS/2 users to print both AFP and current applications without additional hardware.

12.15.3 IBM Print Services Facility/2 Version 1.10

Besides the functions of PSF/2 V.1.0 additionally Version 1.1 enables host PSF/MVS, PSF/VM, and PSF/VSE users to direct output and associated resources automatically to PSF/2 via communications and/or LAN-attached OS/2 systems for spooling and subsequent printing.

PostScript as an input data stream is supported in Version 1.1. HP-PCL5 as an output data stream is also supported in Version 1.1.

PSF/2 Version 1.1 enhancements include:

- Server-based printer sharing between the host and the LAN
- Lower cost printer support (3812, 3816, 4019, 4028, 4029, HP PCL4 and compatibles)
- Improved ASCII emulation (ProPrinter II and QuietWriter III)
- JISCII (Doublebyte ASCII) emulation
- Concurrent spooling/printing on a single PS/2
- Multiple printers driven from each PSF/2
- Expanded connectivity which exploits OS/2 Communications Manager.

A Type Transformer will be provided in Version 1.1 which converts fonts in Adobe Type 1 format to fonts compatible with AFP. These fonts can be delivered to the OS/2, VM, MVS, and OS/400 environments.

The IBM Core Interchange Fonts are provided in Adobe Type 1 outline format for use by the Type Transformer. They consist of three type families (Times New Roman, Helvetica, and Courier) which support the International Organization for Standardization (ISO) single byte character sets for Latin-1 thru Latin-5.

12.15.4 Print Service Facility/2 V1.1

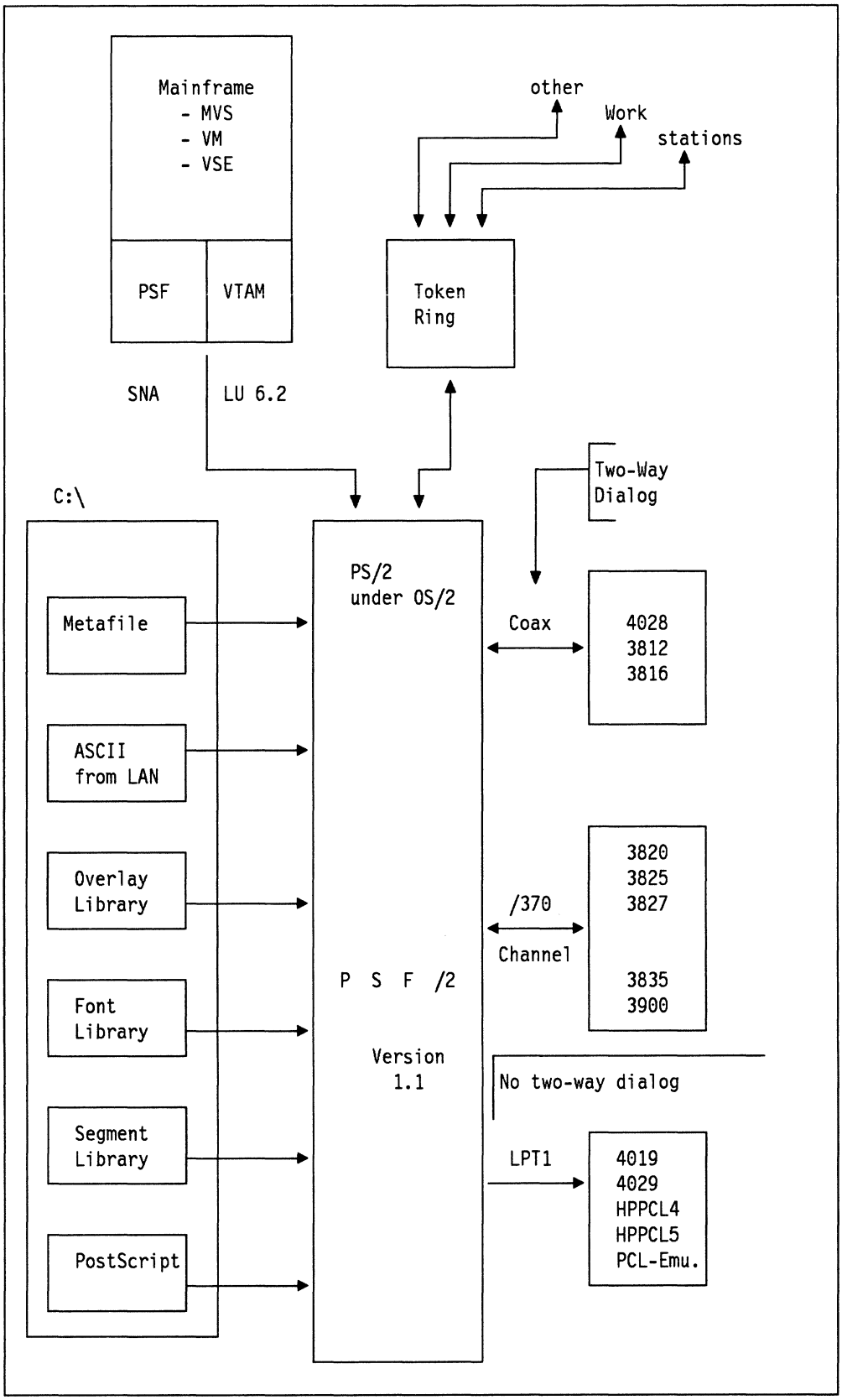


Figure 21. PSF/2 V.1.1 Model for AFP Printing

12.15.5 AFPDS Driver under OS/2 and DOS/WINDOWS

PSF/2 contains both an OS/2 and a Microsoft Windows-based printer device driver that produces Advanced Function Printer Data Stream (AFPDS) output. These are called the IBMAFP printer drivers.

They can be used with any OS/2 or Windows application to produce output printable on AFP printers.

Other versions of these printer drivers are available in the AFP Workbench for Windows product, and internally within IBM as the WINAFP driver.

For image, all of these at present produce IOCA, both compressed and uncompressed, depending on the target printer. For printers that are not capable of accepting IOCA only, PSF V.2. converts uncompressed IOCA into IM, which can then be printed.

The default output mode is to produce a print file, which is made up of a form definition and a document.

Overlay and Page Segment objects can also be produced. They will also contain IOCA, either compressed or uncompressed.

The WINAFP driver can be installed using the Windows control panel. Choose the printer icon, and then install an "Unlisted Printer." You'll then have to download WINAFP.MR0 into the \WINDOWS\SYSTEM directory by hand.

Users of the IBMAFP drivers should refer to the instructions they received with PSF/2.

AS/400 users can use the PC Support/400 Virtual Print function (data type 5) to seamlessly upload the output to the AS/400 spool.

The output contains an inline form definition called IBMAFP. This contains information on duplexing and source drawer. If your PSF platform is unable to use this form definition, the duplexing and drawer selection controls from the main setup dialog will not affect the output.

12.16 Data Streams and Hardware Connection of Printers

Table 4 (Page 1 of 2). Data Streams and Objects by Printer. This table indicates which printers support what Object Content Architecture, and how they are physically connected to a system.

Printer	Supported Data Stream	Connection	Remarks
3900 (with AFIG)	*PTOCA IOCA GOCA FOCA*	*Channel*	*The data streams are enveloped into MO:DCA or into AFPDS and converted to IPDS by PSF or PSF/2 or Print Services/400.*
3835 (with AFIG)	*PTOCA IOCA GOCA FOCA*	*Channel*	*As 3900*
3828	*PTOCA FOCA*	*Channel*	*As 3900*
3827 (with AFIG)	*PTOCA IOCA GOCA FOCA*	*Channel*	*As 3900*
3825 (with AFIG)	*PTOCA IOCA GOCA FOCA*	*Channel*	*As 3900*
3820	*PTOCA FOCA*	*Channel V24, SDLC*	*As 3900*
3816	*PTOCA IOCA GOCA FOCA BCOCA*	*Coax Twinax*	*As 3900; 3816-01S prints simplex and 3816-01D prints duplex*
3812	*PTOCA IOCA GOCA FOCA BCOCA*	*Coax Twinax*	*As 3900*
4028	*PTOCA IOCA GOCA FOCA BCOCA*	*Coax Twinax*	*As 3900; Printer is driven either by PSF, or PSF/2, or Print Services/400.*

Table 4 (Page 2 of 2). Data Streams and Objects by Printer. This table indicates which printers support what Object Content Architecture, and how they are physically connected to a system.

4224	*PTOCA GOCA FOCA BCOCA*	*Coax Twinax*	*certain models support printing of IPDS data stream*
4230	*PTOCA GOCA FOCA BCOCA*	*Coax Twinax*	*certain models support printing of IPDS data stream*
4234	*PTOCA GOCA FOCA BCOCA*	*Coax Twinax*	*certain models support printing of IPDS data stream*
*Lexmark** printers*	*PTOCA IOCA GOCA FOCA BCOCA*	*connected to workstation as LPT1 IPDS is converted to either PPDS or PCL Printer is driven by PSF/2*	*AFPDS printed via AFP Workbench for Windows*
Complementary printers	*PTOCA IOCA GOCA FOCA BCOCA*	*connected to workstation as LPT1 IPDS is converted to either PPDS or PCL Printer is driven by PSF/2*	*AFPDS printed via AFP Workbench for Windows*

Note:
Printers working with the AFIG feature have additional support for IOCA and GOCA available.

Note:
If a channel attached printer is connected to a workstation, a channel adapter card is required.

Note:
If a normal coax or twinax attached printer is connected to a workstation, a coax adapter card is required.

Note:
If a channel attached printer is connected to a workstation, it can be driven either by PSF/2, or RPM V2.0, or RPM V3.0

Chapter 13. Practical Tasks

The purpose of this section is to use practical examples to introduce the relationships between major data streams and objects, and what common programs can be used with them.

It is *not* intended to replace the *ITSC Document Transforms Cookbook, GG24-3530*, to which the reader is referred for details of many transforms not covered in this document.

13.1 Getting Pictures into IBM BookManager from Corel Draw

Corel Draw is extremely useful for both preparing graphics to put into IBM BookManager books, and also for converting graphics from other sources for the same purpose.

Depending on the form of graphics required at the end of the process, however, and on what host software is available, the route taken to convert the graphics will vary, for best results.

13.1.1 Color Pictures

Input The graphic is held in any form valid for Corel Draw on the PC. In this case, we want to use it as a color vector graphic inside IBM BookManager.

The translation process to GDF maps the almost continuous color capability of PC programs to a restricted number of colors. The way these are represented may vary between host graphics terminals and PC color displays.

It is difficult to give general guidelines, and the best advice is to try samples, and build small test books to check rendition by the different routes. If your books are to be used on both host and PC platforms, check on both; rendition will probably be slightly different.

The consistently best results seem to be given by the PIF process out of Corel Draw, and this is the route we recommend you try first.

In any case, where graphics are to be created specifically for use with IBM BookManager, it is recommended that you use colors that you know your translation process will render effectively. A little experimentation the first time will pay off.

The first process below will also work with any other product that can save vector graphics as CGM, but works best with those for which GDDM has color translation profiles.

The profile name quoted below is specific for Corel Draw. Others are:

CGMFP2 For Freelance Plus version 2

CGMFP3 For Freelance Plus version 3

CGMHG For Harvard Graphics**

CGMMD For Micrografx Designer**

CGM General purpose

The second process will only work for PC programs that can save files in IBM PIF format, as Corel Draw can. Where this is possible, this process is to be preferred, as the translation is rather more direct, and color is often rendered rather better on both mainframe and PC platforms.

Process

1. The graphic can be exported from Corel Draw as a CGM, and uploaded in binary from the workstation.
2. Once on the host, the file can be converted to an ADMGDF for direct inclusion in IBM BookManager by one of the following commands:

   VM Use the base GDDM utility

   ```
   ADMUCG filename CGM filemode
          filename ADMGDF filemode
          ( PROFILE CGMCD
   ```

   where "fm" is the disk filemode (usually "a").

   MVS Use the base GDDM utility

   ```
   CALL 'SYS1.GDDMLOAD(ADMUCG)'
        FROM(userid.filename.CGM)
        TO(userid.filename.ADMGDF)
        PROFILE(CGMCD)'
   ```

Alternatively:

1. The graphic is exported as a PIF from Corel Draw, and
2. Either on

   MVS Uploaded in binary, and then

   ```
   ADMUFILE PUT 'userid.filename.ADMGDF'
                'userid.filename.PIFBIN' options
   ```

   or on VM uploaded and converted simultaneously using

   ```
   SEND filename.PIF filename ADMGDF filemode
                                    ( ADMGDF
   ```

Output The output ADMGDF file can now be included in IBM BookManager books, using either the .im macro in GML Starter Set, or the :artwork tag in IBM BookMaster.

Here is an example of a simple graphic which has gone through this second process — it will appear as color in IBM BookManager when displayed on a device supporting color images, but monochrome when printed.

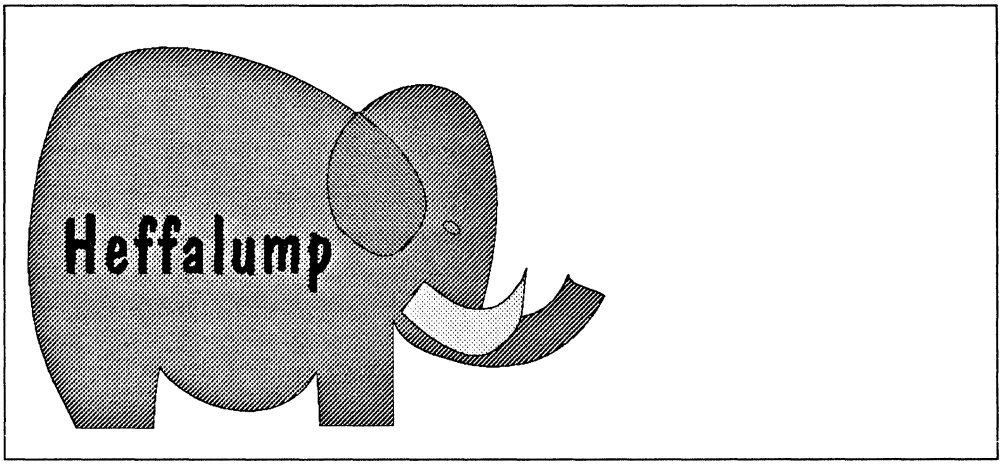


Figure 22. ADMGDF Vector Graphic from Corel Draw

Software required for conversion:

- Workstation
  - Corel Draw
- Host
  - GDDM

13.1.2 Monochrome Pictures

Input The graphic is held in any form valid for Corel Draw on the PC. In this case, we want to use it as a monochrome bitmap graphic inside IBM BookManager.

If, instead of this route, we took the same route as for a vector graphic, via ADMGDF, and then used GDDM to convert the ADMGDF to a PSEG, we would find that all colors in the original, and all shading, would be rendered as black.

For example, this picture:

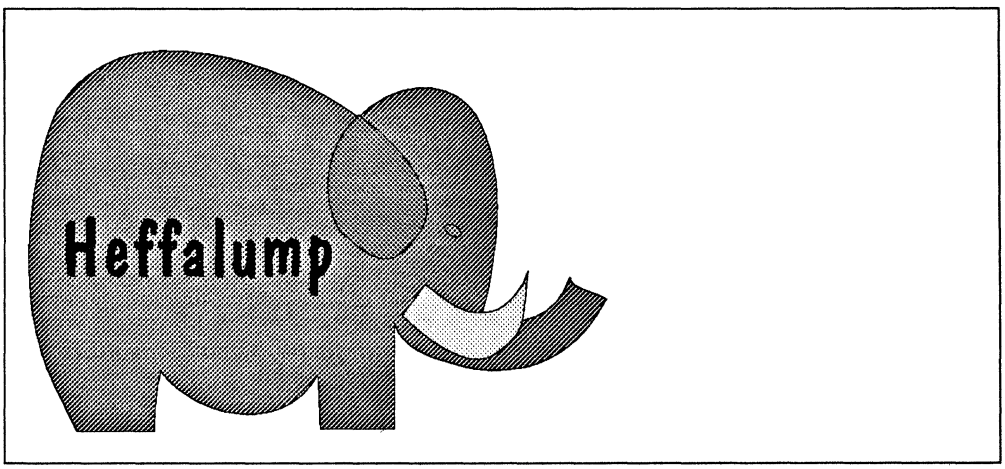


Figure 23. ADMGDF Vector Graphic from Corel Draw

would be rendered as a solid black silhouette of an elephant, while the result we might well prefer, which this procedure will give us, is as follows:

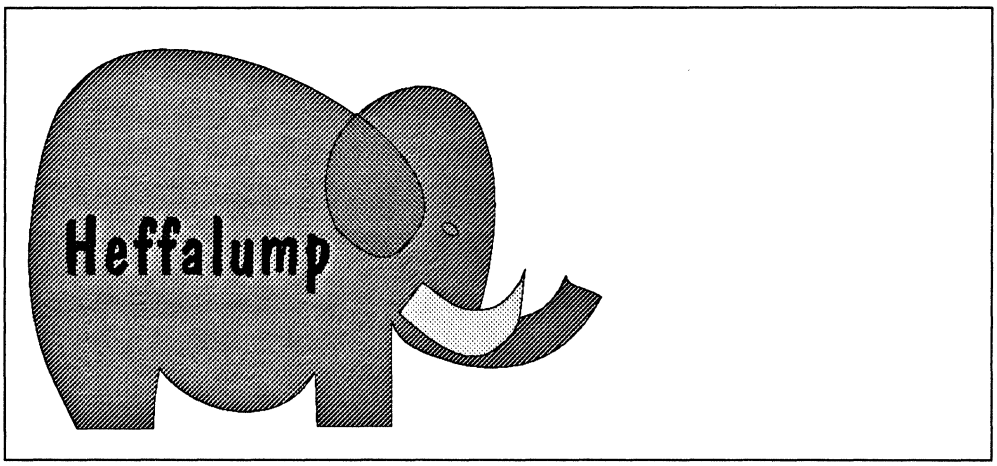


Figure 24. PSEG Created from Corel Draw Graphic

Note: On some output devices, the PSEG might look like the vector graphic, but they are different, believe us!

"All black" works fine if we have a simple line drawing to start with, and we were not concerned with rendering the differing colors as different gray shadings. If this is the case, then this path, using the PostScript Interpreter, is not necessary, and you can use the procedure outlined in 13.1.1, and then use the standard GDDM command on your system to create a PSEG from an ADMGDF. (Use IBM ProcessMaster if you have it, or the ADMUCIMV exec with GDDM, or the ADMUCIMT clist. If in doubt about any of these "standard" transforms, see *ITSC Document Transforms Cookbook, GG24-3530*. Additionally refer to C.24, "ADMUCIMV" on page 207 to see the coding of the EXEC for comparing if your parameters are all correct or where to change if necessary.

Process

1. The graphic should be exported from Corel Draw as an EPS, and uploaded in binary from the workstation.
2. Once on the host, the file can be converted to a PSEG for direct inclusion in IBM BookManager by use of the PostScript Interpreter. Consult your local system documentation for how this is used in your environment — usually this is via a front-end panel.

   Note: If your installation uses the standard EXEC PS370 panel, then ensure the option O is specified for Origin translation, and Y for automatic scaling, in the parameters panel. Provided these are so set, then the PSEG you produce will fill the area you define for the output PSEG in the first panel. Don't make this area too large, or the graphic will take up a lot of space; equally, if you make it too small, then you will lose detail.

Output The output PSEG file can now be included in IBM BookManager books, using either the .im macro in GML Starter Set, or the :artwork. tag in IBM BookMaster.

Software required for conversion:

- Workstation
  - Corel Draw

- Host
  - GDDM
  - PostScript Interpreter

13.2 Combining Desk-top Publishing with GML

In my organization there are several departments that want to create simple documentation with desk-top publishing programs, as they are simple and easy to use with little training.

However, some of this documentation needs to be included into more corporate documents, which are produced using GML.

Is it possible to reconcile the two? Surely we don't have to key the text twice? Is there some way we could convert the formatting done in the desk-top world into GML?

Description of working steps

1. Save your Ami Pro or Ventura Publisher file as ASCII with tag information. With Ami Pro, save files as ASCII while "keeping styles"; this gives a file which can be very easily converted into GML. Ventura Publisher saves its files with tags which can be considered to be at least partially structural; those controls which are format-based can be used to give context for further conversion.
2. Both of these Ami Pro and Ventura Publisher files can be easily converted into GML.

   The reverse process into Ami Pro is just as easy, importing the result of conversion to Ami Pro ASCII format resulting in a document that contains sufficient style information to print directly.

   A sample program capable of conversion in both directions is given in A.1, "Conversion Between Ami Pro and GML" on page 147.

An alternative means of conversion is to use TextTagger if it is available, from a formatted ASCII file provided by either. However, since TextTagger makes its decisions purely on the format of the text, and the tags saved by either of these programs contain more information about document structure than can be inferred from the structure, a transform such as that described in the appendix is preferable in this case. Nor can TextTagger make the reverse transform, which this program can.

However, if the software you are using cannot save tag information, or you have many historical files that are just purely formatted ASCII (or indeed EBCDIC), then TextTagger may well be an appropriate means of getting them into GML.

Requirements Required software

- Workstation
  - Ami Pro or Ventura Publisher, and
  - a simple conversion utility outlined below.
- Host
  - GML Starter Set or IBM BookMaster

13.3 I Want an AFP Print of a Scanned Image Saved as an IOCA

I use OS/2 Image Support to edit and manipulate images. I can save them from this as IOCA files — how do I get PSEGs from that to print in my AFP files?

Scanned images are required in print more and more. The preferred method of converting such images to PSEG format used to be with the IHF product, but this, though excellent in function, is now less often used. IHF remains an option that should be considered by professionals who require high quality output and high functionally in image processing products; it is, however, perhaps less appropriate where simple image transfer is what is required.

For such situations, OS/2 Image Support is an excellent product, and provides as much and probably more functionally than is required in the majority of cases. However, the IOCA files it saves need just a little conversion on the host before they can be used.

Description of working steps

1. The IOCA file must first be uploaded to the mainframe. Specify a FIXED record format, and a record length of 80.
2. GDDM can then be used to convert the IOCA file to a PSEG. A sample exec is given below for VM.

```
/************************************************/
/* Convert IOCA file to a PSEG using GDDMREXX    */
/************************************************/
trace o
arg ifn ift ifm ofn oft ofm

if .||oft=. then oft='PSEG3820'
if .||ofn=. then ofn=ifn
if .||ofm=. then ofm='A'

bpseg='5A0010D3A85F000000D7C7E2F0F0F0F0F0'x     /* afp page segment */
epseg='5A0010D3A95F000000D7C7E2F0F0F0F0F0'x     /*       structures */

/************************************************************/
/* read file into variable rec., number of records in nrecs */
/************************************************************/
'STATE ' ifn ift ifm
if rc¬=0 then exit rc
'FINIS ' ifn ift ifm
nrecs=1
Do forever
  'EXECIO 1 DISKR ' ifn ift ifm '  ( VAR ' rec.nrecs
   if rc¬=0 then leave
   nrecs=nrecs+1
End
nrecs=nrecs-1
/************************************************************/
/* load into GDDM, as image number 1                        */
/************************************************************/
'GDDMREXX INIT'
address GDDM
'IMAPTS  1  0 2   0' /* imageid=1 projid=0 format=2 compression=0 */
do i=1 to nrecs
    'IMAPT 1 . .rec.'i
```

```
    end
'IMAPTE 1'

/*************************************************************/
/* get the image back from GDDM, in PSEG format. put it in rec.*/
/*************************************************************/
'IMAGTS 1 0 3 4' /* imageid=1 projid=0 format=3 compression=4  */

/* build up datastream in rec., counted by nrecs */
nrecs=0;
nrecs=nrecs+1;
rec.nrecs=bpseg    /* add BEGIN PAGE SEG to the datastream*/
do forever
     'IMAGT' 1 2000 ".line .retlen"
    if rc¬=0 | retlen=0 then leave
    nrecs=nrecs+1
    rec.nrecs=left(line,retlen)
    end
'IMAGTE 1'

nrecs=nrecs+1;
rec.nrecs=epseg    /* add END PAGE SEG to the datastream */

/*************************************************************/
/* write out the file from variable rec.,                    */
/*************************************************************/
address command
'ERASE' ofn oft ofm
Do i=1 to nrecs
  'EXECIO 1 DISKW ' ofn oft ofm ' 0 V ( VAR REC.'i
End
'FINIS' ofn oft ofm
exit rc
```

Requirements for conversion

- Software on the workstation:
  - OS/2 Image Support
- Software on the host:
  - GDDM
  - GDDMREXX

13.4 Getting Formulas into Mainframe Publishing

I have an occasional need to create mathematical formulas into documents published on the mainframe, but not so many as to justify SMFF (Script Mathematical Formula Formatter). How can I best create and modify them?

Description of working steps

If you are creating any significant quantity of formulas for the mainframe, and are using GML, then SMFF is probably the best, and certainly the most consistent, method to use.

However, many people have the need for occasional use of formulas, but not enough to justify SMFF. For them, there are several ways of attacking the problem.

1. The simplest, but perhaps the least satisfactory method, is to use a graphics program to draw the formulas, and create a graphics image which can be included in the document. Corel Draw is a program capable of doing this sort of work — but this type of program was not really designed for this purpose, and you would almost certainly have to generate some of your symbology yourself.

2. Some of the more capable desk-top publishing programs and word processors have direct support for formula creation and editing. For example, both Ami Pro and Ventura Publisher have good simple equation editors capable of handling much of this kind of work. The finished product can be exported as an Encapsulated PostScript file into mainframe documents as outlined in 6.1.1, "PostScript" on page 41. This kind of editor has the advantage that if changes need to be made, they are easier to make than with a simple graphics editor. If you have the need for a desk-top publishing program as well, this could be a good solution to the problem.

3. There are dedicated equation editing programs available, and most of these can create PostScript output; so if you have the need for a significant quantity of sophisticated formulas, these are another possible route. However, if your need is such as to be able to justify such an editor, you are also probably a large enough user for SMFF to be a viable solution, and you should consider whether you really need a workstation solution, in which case go for the dedicated editor, or whether your problem is really an enterprise or production publishing one, in which case you should really be using SMFF.

4. One other option not often considered, especially in organizations with academic contacts, is that TEX has reasonable equation handling facilities; and a TEX expression of an equation, though somewhat lower level than a SMFF expression, and rather more format-related, does have some of the advantages of a GML-like equation definition. TEX can be used to create PostScript, where an appropriate filter is part of the TEX installation (usually called *dvi2ps* or something similar), and again this could be included as a graphic. 11.7, "TEX and LATEX" on page 73 outlines the process of taking PostScript from TEX to GML image inclusion.

5. A combination of the equation editor and TEX route is possible with Ami Pro, which allows of WYSIWYG editing of a TEX formula. If Ami Pro is available, then this is probably the route we would recommend.

   This TEX source was created by the Ami Pro equation editor, which is an excellent WYSIWYG tool.

```
$\int\nolimits_a^bf\left( x\right) dx=\lim \frac{b-a}n\sum_{r=1}^n\left(
a+\frac rn\left( b-a\right) \right) $
```

Figure 25. Equation TEX Source from Ami Pro

When saved as a EPS file (the whole page must be saved, and then cropped using the IBM PostScript Interpreter), the following result was obtained:

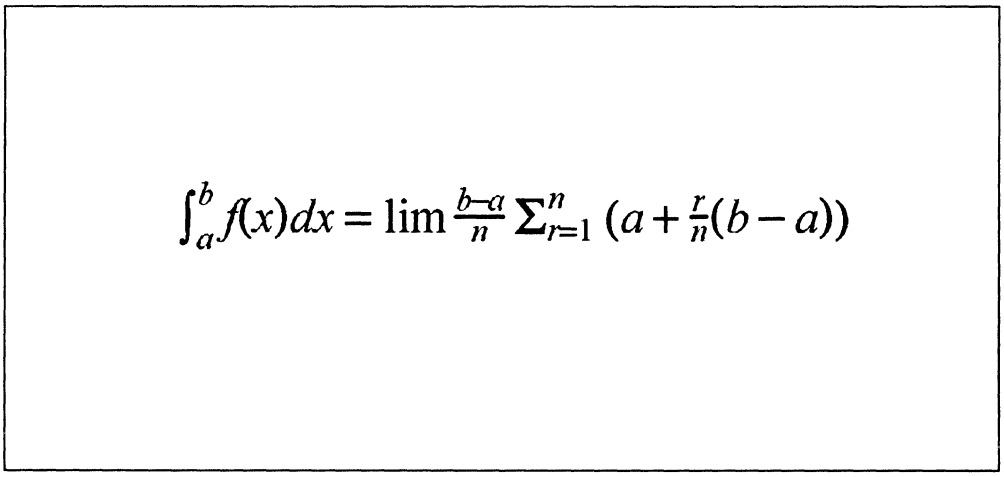


Figure 26. Equation Produced using Ami Pro Equation Editor

(This is using a large font — variable sizes are available.)

(To crop using the IBM PostScript Interpreter, specify the final size you want the PSEG to be, and use the origin and translation option parameters to define where the conversion is to start.)

Requirements Required software:

- Workstation

  Possibly one of the following:
  - Ami Pro
  - Corel Draw
  - T_EX
  - Ventura Publisher
- Host
  - Possibly IBM PostScript Interpreter
  - Possibly SMFF
  - GML Starter Set or IBM BookMaster

13.5 Scan a Logo, Improve It, and Integrate It in an Overlay

I have a company logo in hard copy. It needs to be made into an overlay for AFP.

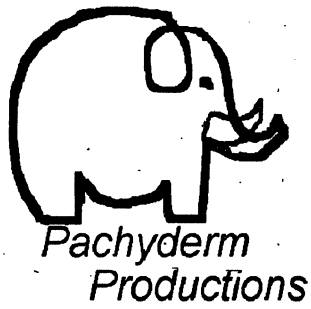
It can be scanned on a PC, but given the resolution of the scanner, the print size would be wrong at 240 pels per inch; also scanning gives a fuzzy image.

What is the best way to get a good logo as an overlay?

Description of working steps

A scanned image can be made into a page segment overlay, cleaned up, at any required size.

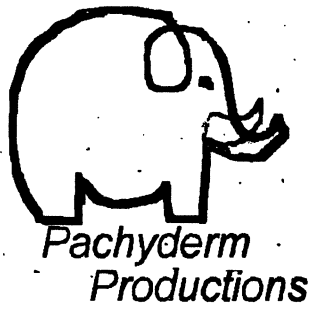
1. First, the logo is scanned using appropriate software for the available scanner, and saved as a TIF. At this stage, the image would look something like the following:



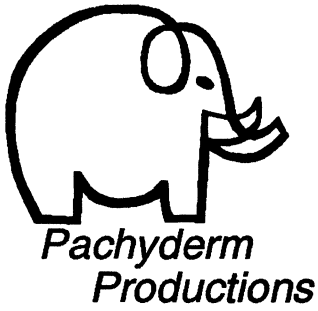
2. The TIF image must then be converted to a vector format, so that the graphic can be resized to any desired ratio without unwanted artefacts such as stepping or fringes. This is done using the Corel Trace program, which is part of the Corel Draw package. In this case we converted it with the "outline" option, which the Corel Trace program then saves as an EPS file, which can be loaded into Corel Draw for cleaning up and conversion.

   Note: *Don't* try to convert halftone images in this way! You'll end up with the largest files in the known universe, as Corel Trace will try its little best to draw vectors round every dot.

   After this stage, the image doesn't look much different:

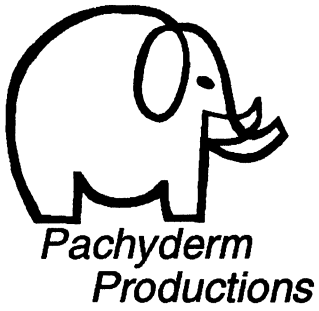


3. Once in vector form, the graphic can be resized and cleaned up using standard functions of Corel Draw, and then saved as an EPS file. Here is the result of that stage — any stepping from here on is the result of printing at a given resolution — the object is now defined as smooth curves:



   If required, the graphic can be freely amended at this stage; changed to any size, or amended. As an example, we include here a new version where we changed the elephant from an

Indian to an African — the ears have been extended — which only took a few seconds.

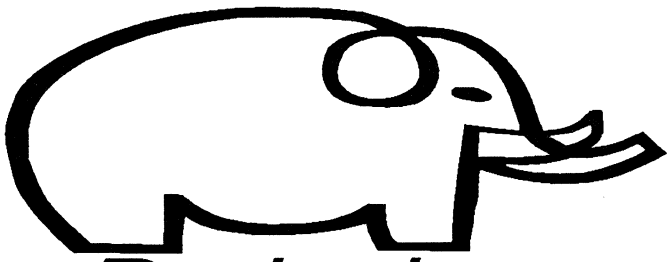


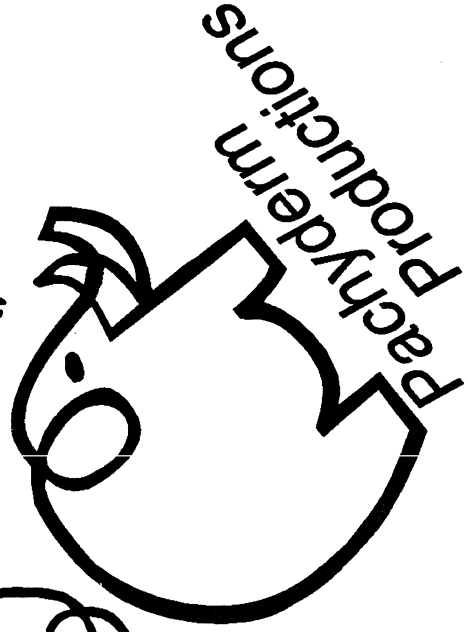
4. The EPS file is uploaded to the host in binary, and then converted to a PSEG using the IBM PostScript Interpreter.
5. Once we have a PSEG, we design the overlay using DCF input to lay out the design of the overlay page, including the PSEG at an appropriate point using either a .im macro or an :artwork. tag, and use this to create an overlay as outlined in 14.5, "Generate Overlays Using the DCF Post Processor" on page 131.

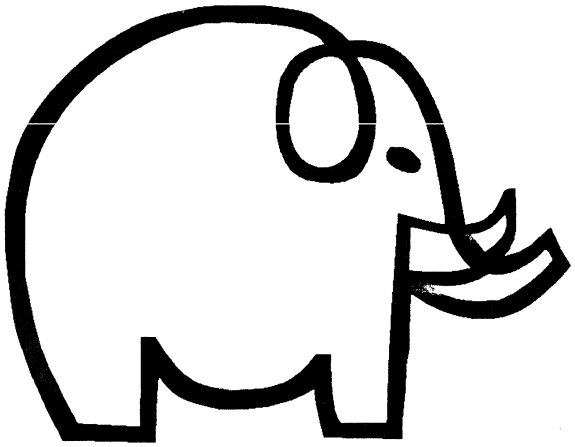
An alternative procedure, once the graphic is in Corel Draw, is to use the IBMAFP printer driver from inside Corel Draw to "print" the image to a file directly as a PSEG. The user is given options at that time to define the size of the resulting PSEG. The only concern here is to make sure that the file is uploaded with the correct file format and record length, and to use the appropriate EXEC on the mainframe to reblock the variable length records. We include a sample exec in A.5, "Reblock Uploaded PSEG" on page 170.

Requirements

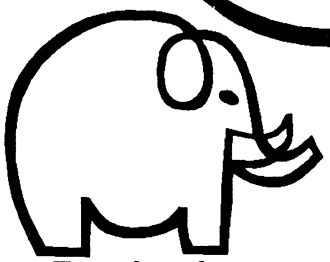
- Required software on the workstation:
  - Scanning Program
  - Corel Trace (part of Corel Draw package)
  - Corel Draw
- Required software on the host:
  - Source Editor
  - OGL





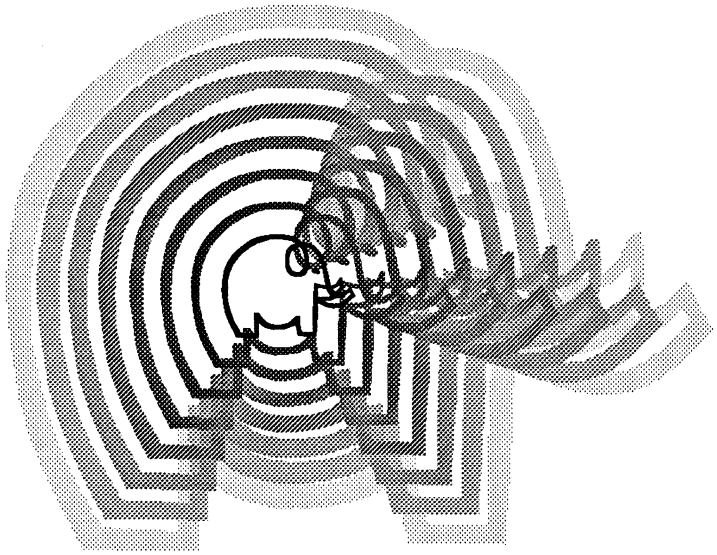


Pachyderm
Productions



Pachyderm
Productions

Pachyderm
Productions



Pachyderm Productions

13.6 I Want an AFP Print of a Scanned Photo Saved as a TIF

I have a scanner attached to my PC. It lets me scan both bilevel images and halftones. The software that came with it saves the images with a filetype of TIF.

How can I get a photograph scanned with this printed on a mainframe AFP printer in a document created on the mainframe in either GML Starter Set or IBM BookMaster?

The TIF object definition is given in 10.3.

Description of working steps

1. The TIF file must first be converted to a PSEG on the mainframe. Some ways this can be done are outlined in 14.3, "Convert TIF to PSEG" on page 130.
2. The PSEG is included in the GML document; if the document is GML Starter Set then an .im macro is used; if it is IBM BookMaster, then an :artwork. tag is used.
3. The document is formatted using DCF, and the resulting output printed on an AFP printer, using a standard system CLIST or EXEC.

Output The output of DCF can be an AFPDS data stream.

Other options for output from DCF include PostScript, described in 6.1.1, "PostScript" on page 41.

Requirements for conversion

- Software on the workstation:
  - Scan program
  - Corel Draw
- Software on the host:
  - GDDM
- Special hardware
  - Scanner, attached to Workstation

13.7 I Have a Plot File and I Want It Printed on the Host

The file is either one created on a Workstation or on the host and needs to be printed in both PostScript format and in AFPDS format.

"Plot file" is a rather generic term. Normally what is meant by this is the HPGL format, though there are other formats defined by other manufacturers. The HPGL format is rather more common, though, particularly in the workstation world.

The HPGL object definition is given in 10.1, "Hewlett-Packard Graphics Language" on page 61.

Description of working steps

- Generated on a workstation
  1. One method of creating first a PostScript, and then a PSEG,

file from a HPGL file is given in 14.4, "Convert HPGL to PSEG" on page 130.

2. If the PSEG is to be printed using AFPDS, it is then included in a GML document or in an overlay.

   Another method is to use the AFPDS printer driver from Corel Draw after importing the HPGL into Corel Draw, as outlined in 13.5, "Scan a Logo, Improve It, and Integrate It in an Overlay" on page 111.

   If the document is GML Starter Set then an .im macro is used, or if it is IBM BookMaster, then an :artwork. tag. The document is formatted using DCF, and the resulting output printed on an AFP printer, using a standard system CLIST or EXEC.
   If the graphic is required to become part of an overlay, it needs to be defined and positioned in the overlay with the IFPDS DEFINE SEGMENT and PLACE SEGMENT commands. The procedure is outlined in 14.5, "Generate Overlays Using the DCF Post Processor" on page 131. The machine-readable overlay then is defined in a FORMDEF, which tells Print Services Facility or Print Services Facility/2 what overlay to use and where on a page to position the overlay.

- Generated on a mainframe

  1. If the plot file is an output file of the CAD/CAM world, GDQF is used for conversion and if necessary for manipulation.
  2. GDQF can be used to create an AFPDS data stream, and IBM ProcessMaster contains conversion facilities to convert this to PostScript, which can then be used on the work station after downloading.

Requirements

- Required software on the workstation:
  - Plot program
  - Corel Draw
- Required software on the host:
  - GDDM ICU
  - GDQF
  - IBM ProcessMaster
  - Overlay post processor of DCF
  - CAD/CAM Software

13.8 I Want to Prepare Host Output for Workstation Print and View

I have host programs that have been used for some years which create line data. I would like to have this output to be able to be printed and viewed on workstations.

What is the best way to do this, and what software do I need either in the workstation or on the host to convert the data?

PSF/2 has the capability of driving many workstation printers from AFPDS, PostScript, and ASCII data streams. If the output data is already in AFPDS or PostScript, then there can be little argument that PSF/2 is the only tool that is required. Equally, if the output can be easily transformed into AFPDS, the same is true.

However, if PSF/2 is not available, or the data stream is one which PSF/2 cannot at the moment transform, then another route must be found.

If the data stream needs to be viewed on the workstation, then AFP Workbench for Windows can be used to:

- Look at the data before it is printed
- Split particular pages out for printing or further use
- Print directly on non-PSF supported printers (IBM Graphics Printer, for example — as a DOS/Windows program, AFP Workbench for Windows can print on any output device for which you have a driver)
- Transformed into a PS or PCL file — again, any filetype for which you have a driver

Options: There are basically three data streams that are commonly used on the workstation for printing:

- PPDS
- PostScript
- PCL

Which is to be selected depends on the capabilities of the printers you have available, and the format of the output data that is to be printed.

PPDS is most appropriate where the printer population is essentially IBM workstation printers or non-IBM devices which use IBM controls.

PCL is supported by some IBM workstation printers and several non-IBM suppliers.

PostScript is supported generally only by higher function printers, but has the advantage of device independence, richness of function, and the capability of also being used as a view data stream with the faster workstations using Display PostScript, which in general PPDS and PCL cannot. PostScript printers are also becoming more cheaper and more common.

There are no specific conversion products available to turn line data with mainframe carriage controls into these data streams; however, there are several utilities available on the PCTOOLS disk to convert such 1403 data files to PPDS. In any case, simple print programs can be written to do the task.

Alternatively, word processor programs can often import plain ASCII data, and can be used to print directly, or after minimal formatting.

So, the best solution seems to be to try to get the data stream from the mainframe already in AFPDS form. In that form, you then have the fullest flexibility for workstation use as well as host use, given the availability of PSF and AFP Workbench for Windows.

Chapter 15, "Transforms" on page 135 deals with general considerations about transforms of all kinds, and the three data streams above are outlined in

- 5.1.6, "IBM Personal Printer Data Stream" on page 36
- 6.1.1, "PostScript" on page 41
- 7.2, "PCL (Printer Control Language)" on page 45

13.9 How do I Use PSEG Files with my DTP Programs?

I have many graphics on the mainframe in PSEG form, and in many cases the original source is long gone. I want to be able to use them in workstation programs. How do I get them there without printing them and scanning them fresh?

These graphics need to be converted to TIF, as being probably the most popular image format for use with DTP programs on the workstation.

Once converted to TIF, then can then be used, edited by many programs, or even, if they are not halftoned or otherwise composed of very many small elements, converted to a vector format by such a program as Corel Trace.

Description of working steps

1. The first step is to convert the PSEG to a form that can be handled by OS/2 Image Support — the preferred form is IOCA. A PSEG can be converted to IOCA either by using GDDM IVU, or by using GDDM functions directly in a similar way to the reverse conversion outlined in 13.3, "I Want an AFP Print of a Scanned Image Saved as an IOCA" on page 108.
2. OS/2 Image Support can then be used to convert the IOCA files to TIF, together with any appropriate resizing, cropping, and so on.

An alternative option is the use of GDQF. GDQF can load and display a PSEG, and then save it in several formats, three of which are different types of TIF (using different compression algorithms). Test all three of those with the workstation program you want to use, as not all programs can handle all types of TIF compression.

Requirements

- Required software on the workstation:
  - OS/2 Image Support
- Required software on the host:
  - GDDMIVU
  - GDQF

13.10 How Do I Use Mainframe CAD Pictures in DTP?

The host has CAD models stored in a library, and I can access them with GDQF, or create GDF files from the CATIA** or CADAM** models directly. How can these be cleaned up and used in workstation DTP programs?

GDF is an IBM mainframe format; what format is needed on the workstation will depend on the software used. Possibilities are:

- CGM
- PCX
- TIF

and many others. The best approach is to convert them to a form with minimal loss of information, and store them with a program capable of exporting to as many different formats as possible. One possible candidate for this is Corel Draw.

Description of working steps

1. Download the GDF file as a PIF file. This can be done using the PC3270 emulator using the ADMGDF option:

   ```
   receive filename.ext  filename admgdf fm ( admgdf
   ```

2. The PIF file can then be directly imported into Corel Draw, and stored as a Corel Draw file. At this stage, all the artistic attributes such as color are preserved.

3. Corel Draw can then be used to export the graphic in a number of formats, of which TIF is one. Most DTP programs can make use of TIF graphics, or EPS, which Corel Draw also provides.

   Be careful, though, of some of the transforms which may lose information. If you want to retain color, for example, then the options you have may be limited; also, some of the transforms which convert color to monochrome will do a more appropriate job of converting color to halftone patterns than others for some tasks.

Requirements

- Required software on the workstation:
  - DTP software
  - Corel Draw
- Required software on the host:
  - CADAM, CATIA, GDQF, etc.

13.11 How do I Use Output from Workstation Programs on the Host?

Many users in my organization use workstation programs under OS/2 and DOS/Windows to create material, both text and graphics. We have a significant investment in these programs, and much information in the form they use. It would cost too much to switch everybody to some new set of standard programs, both in retraining and software costs.

It would be useful in many cases to be able to incorporate some of this material into mainframe applications. One of the major problems we have is being able to archive all this stuff. It would be useful to be able to search through it all for particular information, too.

How can we incorporate all this workstation information easily?

Graphics are fairly straightforward, as is text or mixed text and graphics output if you are happy to treat them as effectively graphics objects. Printing them on the workstation using the AFPDS driver as described below will produce files that

are essentially IOCA (wrapped up as AFPDS, PSEG, or OVERLAY files). If what you want is a page or other image, this is fine.

The problem with this is that the information content can be printed or displayed, but can't be searched on — the text strings no longer exist; they are bitmaps, which are rather harder to search.

Producing a searchable file that is usable on the host can be difficult from a GUI-based workstation. The only printer drivers that produce content with embedded strings are usually the PostScript or PCL ones, and neither are an ideal input medium; they are designed as Page Description Languages, and are not an easily used data input format. There are generic text printer drivers, but many GUI programs won't work with these; they produce either a PostScript definition, or a bitmap in one form or another. The printer drivers that do include text tend to be like PostScript or PCL, and not to be suited to generic input.

So if the text or formatted data is to be used, the output of workstation programs often can't be used.

Often, the input, stored, or exported version of the data is a better route; some of these are described in 15.2, "Round and Round We Go ..." on page 138. If you can get your data into IBM BookManager in this way, the text is searchable; this is a good way to create an archive of workstation documentation.

Description of working steps

1. To produce an AFPDS file from within OS/2 or DOS/Windows, make sure you have either the PSF/2 OS/2 or Windows printer driver (*IBMAFP*), or the WINAFP printer driver installed. (WINAFP is available on PCTOOLS — the *IBMAFP* printer drivers are part of the PSF/2 and AFP Workbench for Windows products.)
2. Use the DOS/Windows Control Panel to connect the WINAFP driver to a file instead of :LPT1 (or similar functions for the other drivers, and so on below).

   The setup functions accessible from the DOS/Windows Control Panel allow you to specify whether the file to be produced is to be an AFP listing file, a PSEG, or an overlay.
3. Upload to the mainframe; remember to use a file transfer set to variable record format and maximum record length of 32767, and then use the supplied AFPSPLIT EXEC to reformat the file appropriately.

If you use PSEGs made in this way in IBM BookMaster, you will find a slight problem. The driver partitions images into a number of segments which are held together in one file. Unfortunately, IBM BookMaster takes the dimension of the first segment as the space to assign in the document for the whole, and then goes on to use the whole. This means that the PSEG will overflow its assigned space, and if you don't leave space after it, it could wrap over following text, as in the example below.

Here's what we mean; we'll repeat the last paragraph a couple of times for it to wrap over so you don't miss anything. (Oh, and by the way, IBM BookManager users — we haven't gone insane; you won't see any of these problems, so bear with us for a minute.)

(BEGIN OF REPEATED TEXT.)

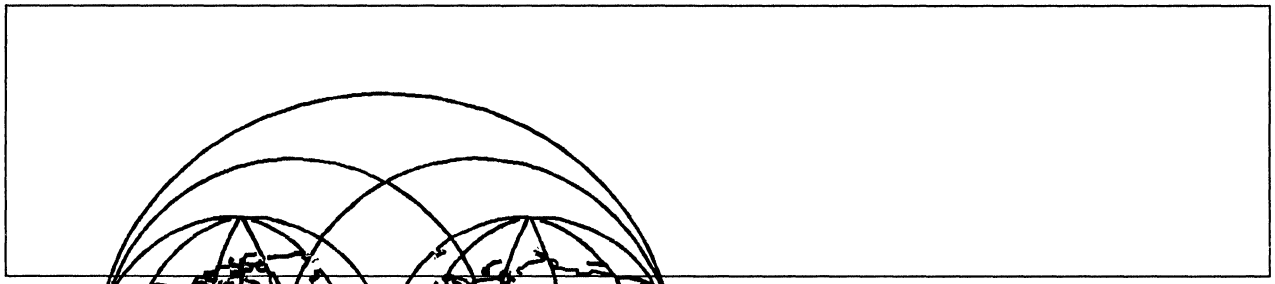


Figure 25. A WINAFP PSEG Overflow

If you use PSEGs made in this way in IBM BookMaster, you will find a slight problem. At the time of writing, the driver partitions images into a number of segments which are held together in one file. Unfortunately, IBM BookMaster takes the dimension of the first segment as the space to assign in the document for the whole, and then goes on to use the whole. This means that the PSEG will overflow its assigned space, and if you don't leave space after it, it could wrap over following text, like the example below.

If you use PSEGs made in this way in IBM BookMaster, you will find a slight problem. The driver partitions images into a number of segments which are held together in one file. Unfortunately, IBM BookMaster takes the dimension of the first segment as the space to assign in the document for the whole, and then goes on to use the whole. This means that the PSEG will overflow its assigned space, and if you don't leave space after it, it could wrap over following text, like the example below.

The overprinting of the picture and text on this page is intentional. It shows what happens if the bottom margin of the picture is not adjusted in the :artwork tag.

(BACK TO UNREPEATED TEXT HERE.) While here we fix it by adding some whitespace to the :artwork. tag, as here:

```
:artwork botmar=3i name=world2
```

instead of

```
:artwork name=world2.
```

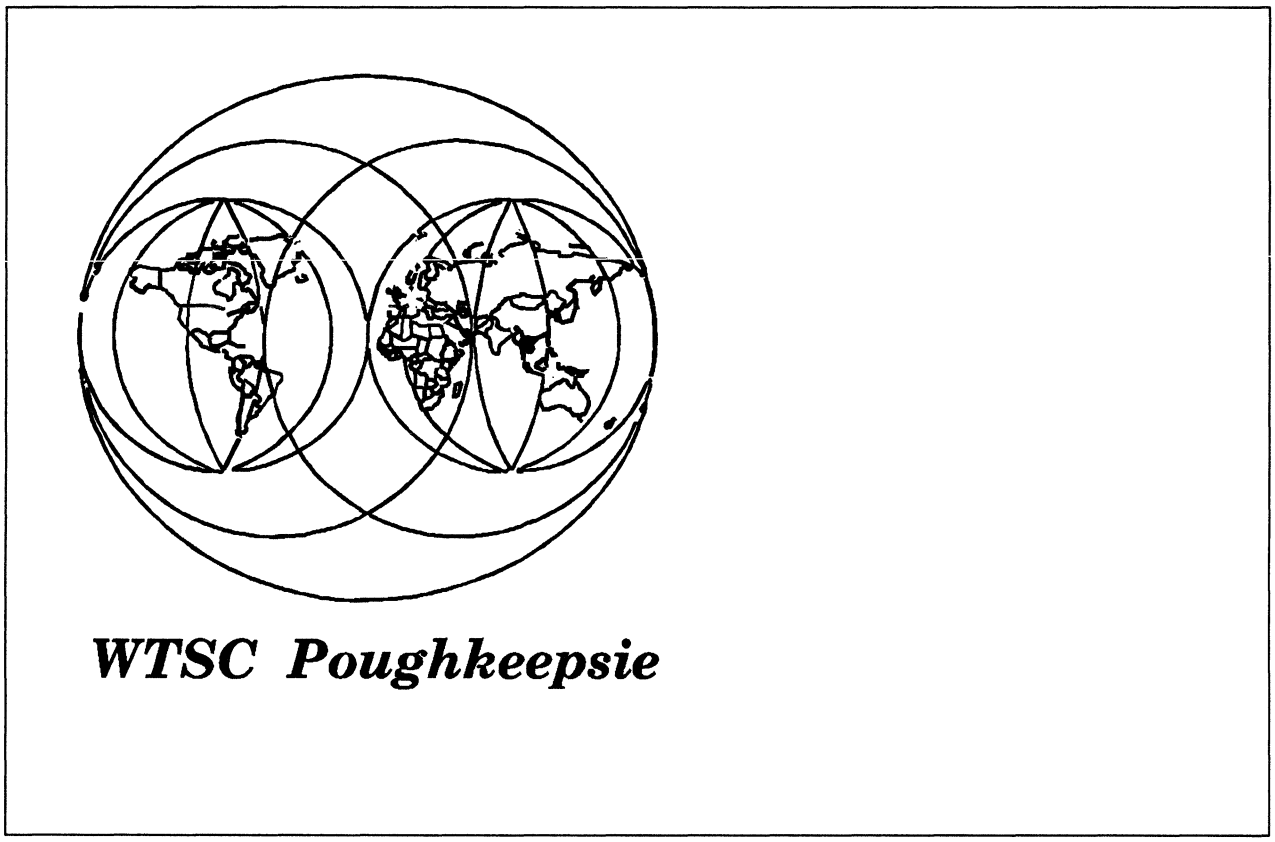


Figure 28. A WINAFP PSEG not Overflowing

You'll probably get messages about the PSEG overflowing the right page boundary, as well — this is because the PSEG extent information is full page width (and adding the IBM BookMaster margin to that makes it overflow). Don't worry about that; it's probably all empty space, unless of course you did make the PSEG with content all the way over to the right hand edge of the page, in which case *go back and make it smaller*.

Note: There is one further step necessary if you want to use this kind of segmented PSEG in IBM BookManager. BookManager BUILD ignores all but the first segment of such PSEGs, so you would only see a small part of them included in the book.

To show you the effect, we left the first picture unconverted. You probably noticed, and wondered what was happening. (This time people reading the paper book won't know what we're going on about, as IBM BookMaster uses all the graphic.) The solution is to recombine the segments. This can be done with a simple REXX exec on the mainframe, and a sample one is included in A.4, "Combine Image Cells in PSEG" on page 169.

By the way, the PSEG we used above has had an interesting career. We found it loitering on a disk as a PSEG, and decided to play. We loaded it into GDQF, and saved it as a TIF file, then downloaded it to a PC. On the PC, we loaded the TIF into Corel Trace, and used that to change it to a vector file. That vector file was then put into Corel Draw, where we added the text "WTSC Poughkeepsie." In Corel Draw, we printed it as a PSEG as described above, and then brought it back to the mainframe.

As it stands, it could use a little cleaning up, but it shows the level of fidelity that can be preserved through that kind of cycle. The vast majority of the degradation happened in the PSEG to vector conversion in Corel Trace, as you would expect. However, once in vector form, cleaning up pictures in an editor like Corel Draw is relatively quick and easy.

We've put the original below so you can compare for yourself.

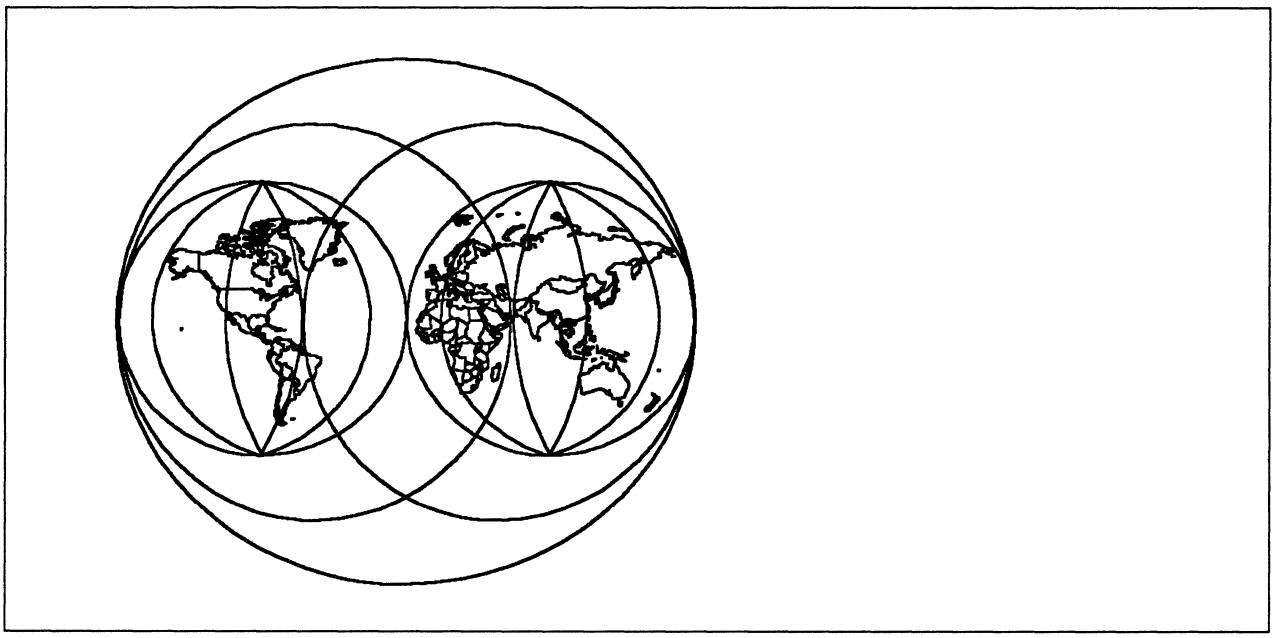


Figure 29. The Original PSEG

13.12 I Want Bigger Volumes of Print from my Workstation Applications

We have working solutions to our business needs using various workstation applications, but the printers aren't fast enough to process the volumes.

The faster IBM printers are AFP — can we get the output from the existing applications printed on them? What do we have to do in order to make it all automated, with the printers on the workstations too? We don't want to have to beef up the mainframe or communications network just to handle the local print load.

PSF/2 version 1.1 has conversion filters built into it. Providing your applications can provide:

- PostScript
- AFPDS

- ASCII
- Metafile

output data streams, then PSF/2 will take that as input, and can convert it to IPDS and print it on fast AFP printers.

If you don't have access to PSF/2, but you want the occasional piece of workstation output to print on an AFP printer for volume or quality, then investigate using the IBMAFP printer driver from AFP Workbench for Windows. Once you produce an AFPDS file, you can upload and print on a host AFP printer, after reblocking the file using an exec like that in A.5, "Reblock Uploaded PSEG" on page 170.

Main requirements

1. Generate appropriate input:
   a. DTP programs are used to produce combined text and drawings which must be printed on one page.
   b. A work station application program generates address data in ASCII.
   c. Text, drawings, and address data are combined for a direct mail campaign.
2. Get the hardware and software in place:
   a. Attach an AFP printer to a workstation in a LAN and have it driven by PSF/2.

Result High volume print; no change to existing procedures to produce the input; minimum additional control needed.

Chapter 14. Process Definitions

This chapter introduces how format conversions between data stream can be done.

14.1 Format Conversions

The data streams shown in Figure 30 on page 129 are amongst the most popular being produced by applications, and there is often the need to convert them into another format for further processing.

Rather than describe all conversions in detail, we have here a list of some of the simpler conversions together with the tools needed to perform them. There are later sections that deal with some conversions that justify more explanation.

We can also recommend *ITSC Document Transforms Cookbook, GG24-3530*, for other methods and conversions not cited.

Convert IOCA to LIST38xx

- Standard function of GDDM-IVU.

Convert IOCA to PSEG38xx

- Standard function of GDDM-IVU.

Convert IOCA to PostScript

- Use GDDM-IVU to convert the IOCA file to a PSEG.
- Imbed the PSEG in a dummy DCF document.
- Define PostScript as the output device to DCF.
- DCF creates a PostScript document.

Convert IOCA to OVLY38xx

- Use GDDM-IVU to convert the IOCA file to a PSEG.
- Imbed the PSEG in a dummy DCF document.
- Define LIST3820 file as the output device to DCF.
- DCF creates a normal document.
- Use the DCF post processor to generate an overlay.

Convert LIST38xx to IOCA

- Standard function of the GDQF image editor.

Convert LIST38xx to PSEG38xx

- Standard function of GDDM-IVU.

Convert LIST38xx to PostScript

Method 1

- Use AFP to PostScript (under ProcessMaster*)

	Method 2 • Use AFP Workbench for Windows on the workstation.
Convert LIST38xx to OVLY38xx	Method 1 • Use the DCF post processor to generate an overlay from a LIST38xx. Method 2 • Use AFP Workbench for Windows on the workstation.
Convert PSEG38xx to LIST38xx	Method 1 • Standard function of GDDM-IVU. Method 2 • Imbed the PSEG in a dummy DCF document. • Define LIST3820 file as the output device to DCF. • DCF creates the document.
Convert PSEG38xx to IOCA	• Standard function of the GDQF image editor.
Convert PSEG38xx to PostScript	Method 1 • Imbed the PSEG in a dummy DCF document. • Define LIST3820 file as the output device to DCF. • DCF creates the document. Method 2 • Use AFP Workbench for Windows on the workstation.
Convert PSEG38xx to OVLY38xx	Method 1 • Imbed the PSEG in a dummy DCF document. • Define LIST3820 file as the output device to DCF. • DCF creates the document. • Use the DCF post processor to generate an overlay. Method 2 • Use AFP Workbench for Windows on the workstation.
Convert PostScript to LIST38xx	• Use PostScript to AFP (under ProcessMaster).

Convert PostScript to PSEG38xx

- Use PostScript to AFP (under ProcessMaster) to create a LIST38xx.
- Use GDDM-IVU to create the page segment from the LIST38xx.

Convert PostScript to IOCA

- Use PostScript to AFP (under ProcessMaster) to create a LIST38xx.
- Use GDQF to create the IOCA file from the LIST38xx.

Convert PostScript to OVLY38xx

- Use PostScript to AFP (under ProcessMaster) to create a LIST38xx.
- Use GDDM-IVU to create a page segment from the LIST38xx.
- Imbed the PSEG in a dummy DCF document.
- Define LIST3820 file as the output device to DCF.
- DCF creates a normal document.
- Use the DCF post processor to generate an overlay.

Convert OVLY38xx to IOCA

- Standard function of the GDQF image editor.

Convert OVLY38xx to PSEG38xx

- Standard function of the GDQF image editor.

Convert OVLY38xx to PostScript

- Include the overlay in a dummy DCF document. The overlay is included with using either the .cg or .oi IBM Script command.
- Define PostScript as the output device to DCF.
- DCF creates the document.

14.2 Image Conversion under GDQF

Since GDQF is mentioned a lot of times as it provides tools to convert from one format to another, we include here an overview of these.

The image conversion utility of GDQF allows the conversion of many types of image objects into other formats.

By providing appropriate options, converted images can also be rotated, sized, negated, scaled or changed in resolution.

GDQF can convert to:

- ADMIMG
- AFPDS
  - PSEG38PP
  - PSEG3820
  - LIST38PP
  - LIST3820
  - OVLY38PP
  - OVLY3820
- Bitmap
- CALS
- IOCA
- MO:DCA
- RFTDCA
- TIFF
  - No compression
  - CCITT G.3
  - CCITT G.4
- CALS

For more information about converting image data please refer to *Graphical Display and Query Facility Using GDQF Base Ver. 2 Rel.1.0 (VM), SH52-0252* when you work under VM.

For more information about converting image data please refer to *Graphical Display and Query Facility Using GDQF Base Vers. 2 Rel. 1.0(MVS), SH52-0253* when you work under MVS

Note also the large number of transforms from AFPDS to other types, which is possible using AFP Workbench for Windows. By diverting the output to a file (an option in the Printer Setup function), you can use this product to create any printer data stream for which you have a printer driver. You have to install these using the "Install Printer" functions, but you will have available to you at least:

- PostScript
- PCL
- PPDS

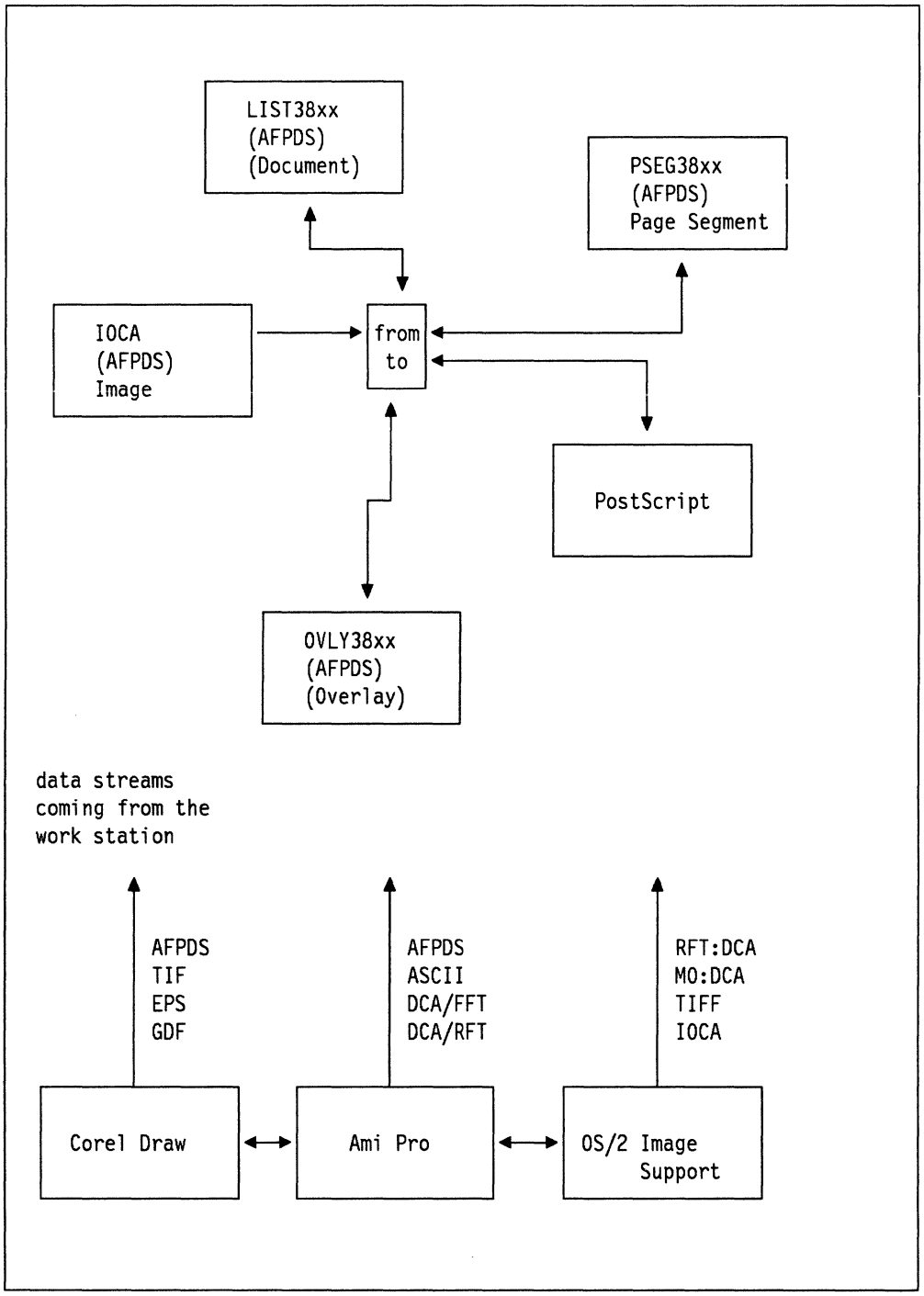


Figure 30. Popular Data Streams and Some Transform Software

14.3 Convert TIF to PSEG

14.3.1 Using DOS Platforms

One method of converting a TIF to a PSEG uses Corel Draw, a PC product described in C.32, "Corel Draw" on page 216. This is probably the preferred route when the PC platform is running PC-DOS, provided that the application supplying the TIF file produces a TIF that is readable by Corel Draw. Note that not all files with a file extension of TIF are necessarily to the TIF specification, and it is possible that some versions of Corel Draw do not necessarily cope with the full TIF specification. However, most applications provide compatible subsets.

1. The TIF file is imported into Corel Draw, and saved as an EPS file, using standard Corel Draw functions.
2. The EPS file is uploaded to the mainframe as a binary object without conversion using standard file transfer.
3. The EPS file on the mainframe is converted to a PSEG using the IBM PostScript Interpreter.

An alternative route is if you have the IBMAFP or WINAFP AFPDS printer driver installed; then printing to a file allows the direct creation of a PSEG. This also needs an EXEC on the mainframe to reblock the file, or an equivalent MVS procedure to that supplied for VM. If such an exec wasn't supplied with your software, you might like to try that in A.5, "Reblock Uploaded PSEG" on page 170.

Publisher's Paintbrush will also convert TIF to EPS.

14.3.2 Using OS/2 Platforms

Corel Draw will run under OS/2 as a DOS/Windows program, and is also available as a native OS/2 application, so can be used under OS/2 to convert TIF to EPS, and then follow the same path as under the PC-DOS method. The IBMAFP printer driver is also available under OS/2.

14.4 Convert HPGL to PSEG

14.4.1 Using DOS

One method of converting a Plot File to a PSEG uses Corel Draw, a PC product described in C.32, "Corel Draw" on page 216.

1. The Plot File file is imported into Corel Draw, and saved as an EPS file, using standard Corel Draw functions. The HPGL file is "imported" into Corel Draw, and will be placed on the page into which it is imported at a position and size determined by the original size and placement definitions inside the plot file. Note that an HPGL file is a defined size, and may be offset from the defined origin.

   Corel Draw can be used to resize the graphic before further transforms. The trap select function using the crunch box can be used to select all the elements of the graphic, which can then be resized and repositioned on the page.

An EPS file is created from the graphic by using the "export" function. Note that there is an option in Corel Draw EPS export to include an "image header." This *must not* be selected if the IBM PostScript Interpreter is to be used on the file. This image header is only used by certain workstation programs to give a preview of the contents of the EPS without having to interpret the PostScript. The image header is *not* standard PostScript, and many programs, including the IBM PostScript Interpreter, will reject the file if it is preceded by an image header.

2. The EPS file is uploaded to the mainframe as a binary object without conversion using standard file transfer.
3. The EPS file on the mainframe is converted to a PSEG using the IBM PostScript Interpreter. The IBM PostScript Interpreter allows resizing and cropping of the graphic. An EPS file contains size definitions; with the IBM PostScript Interpreter, you can define the size of the resulting PSEG, the relative origin inside the EPS that is to be the origin of the PSEG, and a scale factor that determines the relative sizes of elements in the PSEG compared to the original EPS.

If you don't have the IBM PostScript Interpreter, and have access to an IBMAFP printer driver (from either PSF/2 or AFP Workbench for Windows, or the internal WINAFP driver), then once the graphic is in Corel Draw it can be "printed" directly as a PSEG, either under DOS or OS/2. See 14.3, "Convert TIF to PSEG" on page 130 for more information.

14.4.2 Using OS/2

Corel Draw will run under OS/2 as a DOS/Windows program, and is also available as a native OS/2 application, so can be used under OS/2 to convert Plot File to EPS, and then follow the same path as under the PC-DOS method.

14.4.3 Using GDDM on the Mainframe

If the file is on a host system, it can be converted with either GDDM Interactive Chart Utility or with GDQF. Both of these can import an HPGL file and save it as a GDF.

The GDF can then be converted to a PSEG as outlined in 13.1.2, "Monochrome Pictures" on page 105.

14.5 Generate Overlays Using the DCF Post Processor

The overlay should be prepared using DCF input to create a page image of the layout and size required. DCF can then be used to create a LIST3820 file, which is used as input to the Post Processor.

The Post Processor is part of DCF, and is available under:

- VM
- MVS
- VSE

but the Overlay postprocessor is not supplied as a VSE runtime. The DCF postprocessor PL/I source code is supplied as part of the package.

The postprocessor places each page of the formatted document into a separate file. It just has to be run against the DCF input; no other user input is required.

The exec or CLIST may be called *PPOVLRUN*. If you can't find that, talk to your system administrator.

For more information about the Post Processor and its use, refer to *DCF Post Processor Examples S544-3484-00* which also describes a mail merge post processor, and one to print page printer output multiple up (print several pages on one).

14.6 Preparing PostScript

There is an overview of what PostScript and EPS are in 6.1.1, "PostScript" on page 41.

14.6.1 On the Workstation

There are two ways PostScript can be produced on the workstation, usually under a GUI — either DOS/Windows or OS/2.

1. Directly by an application
2. By the GUI using a printer driver

Usually EPS is produced only by the first of these.

There shouldn't be any great difference in the output provided by either method; but usually now applications are making more use of the GUI facilities, and more fonts tend to be available to the applications that do this.

The one warning note about using some of these fonts is that unless you can be sure that the fonts you use are resident in your printer, *and you tell the application or printer driver that this is the case* (usually through an options panel), the more fonts you use the larger will be your files and communications time. Sometimes these can be very large indeed.

In any case, it is generally good practice to restrict the number of fonts used in any single document, unless there is very good reason to do otherwise.

When the GUI has a PostScript printer driver, this means that any application using the GUI, even if it doesn't have native PostScript facilities, can then have its output produced in PostScript

Note: This doesn't apply to DOS programs running under DOS/Windows. These usually talk directly to the printer instead of using the GUI drivers.

Printer drivers, as their name implies, are normally used to spool the output of various programs to a printer, in this case a PostScript printer. It is sometimes the case that we don't want to print directly, but create a PostScript (or Encapsulated PostScript) file to use for further processing, perhaps to be transformed into something else.

This can usually be done by changing an option in the GUI. In DOS/Windows, for example, there is a "connect" option in the *printer setup* function in the Control Panel. This can be used to point the output of any printer driver to any available port on the machine, or to a file.

When you have the printer driver directed to a file, every time you print something you are then prompted for the file name to send the output to.

14.6.2 Using ProcessMaster (VM, MVS)

IBM ProcessMaster has a transform program to create PostScript from any AFPDS data stream. So when IBM ProcessMaster is available, any application that produces AFP output can also be used to generate PostScript.

14.6.3 Using DCF

DCF is used to generate PostScript by defining PostScript as the device type for output. So, any application that uses DCF as a formatter is capable of generating PostScript. That includes:

BookMaster

The SGML Translator

as well as any user written applications, such as database publishing.

14.7 Preparing PCL

This is a typical workstation data stream.

PCL is usually generated by a workstation printer driver. The same comments about printer drivers, and direction of their output to files, as written in 14.6, "Preparing PostScript" on page 132 apply.

For more information about PCL data streams refer to 7.2, "PCL (Printer Control Language)" on page 45.

Chapter 15. Transforms

There are many programs which transform data from one format to another. In some cases, this is as a side issue to the main function of the program — an example is Corel Draw, which can import and export many file formats, and thus transform between them. This function is in Corel Draw so that Corel Draw can use artwork from many sources, and be used in many situations, and not for it to be used primarily as a conversion tool. Nonetheless, it does allow this, if a more direct route is not available.

Some system utilities are more directly aimed at transformation, together with other system function. An example here might be GDDM, which besides being a graphics facilitator, has specific functions for converting file types.

Finally, there are purpose written conversion utilities, which might be hard coded to perform a single transform, or can be tailored to meet a wide range of conversions. An example of the latter is TextTagger.

Even with all of these available, there will be many occasions when the transform required is not available, or if available, doesn't work quite as required, or might be a host program when the change is required on a PC. It is then that a custom transform might need to be written. This chapter aims at assisting in these cases, by pointing out some of the more obvious points that should be considered when choosing which path to take, and how to design and how to write transform programs that take you along that route.

One piece of practical experience is that there is a fine balancing act to be performed between being very specific in the task that is addressed, and aiming for a little more generality. We show in Appendix A, "Source Code for Sample Transforms and EXECs" on page 147 four quite different transforms. Of these, two are quite specific in the tasks they were written to perform. The other two are a little bit more general, in that they do a particular type of job on types of files, but are table driven in the precise functions they perform, these tables being read in from an external "profile" file.

Specific transforms are usually quicker to write, but are clearly relatively inflexible. They are the most efficient at doing a specific job, which can be important for intensive tasks. Where the initial specification is ill defined, or in an environment subject to change, a more generalized approach, whether controlled by a profile or other means, can be more effective not only in the long run, but actually in the quite short term. It is often not that much more difficult to write generalized transforms than complex specific ones, and sometimes simpler. The more generalized transforms not only are more resistant to the effects of a changing environment, but can often be reused for purposes unconnected with the original design.

One point which usually selects itself is the language to write the code in. This will be selected by a combination of factors, such as what skills are available, and what platform the job is to be done on. Most languages will do most jobs, sometimes with a little tweaking, so this is not a very important choice; we have used C and REXX for our examples because they are relatively commonly understood, they are capable, and we happen to understand their basics.

15.1 Types of Transform

A little consideration of the types of transforms, and their implications, can be useful in deciding whether to attempt to try to follow a particular route, and can give an idea of the likelihood of success, and how faithfully the transformed data stream can be expected to reflect the original.

Transforms can be categorized in many ways, but it is useful to distinguish between:

- Reformat transforms
- Change state transforms

An example of a *reformat* transform would be one that changed a formatted text file from one WYSIWYG word processor format to the format used by a similar word processor produced by another manufacturer. All the data would be preserved, the text would remain text, but the formatting controls and commands would be different. In reformatting, the data stays essentially the same; only the form in which it is expressed changes.

In general, reformat transforms are the simplest to write and the most likely to be successful.

An example of a *change state* transform would be one that converted a file of plot commands to a bitmap. The resulting image when printed might look very similar, but a definite change in the way the data is held has taken place; a change that is different in kind from a simple reformat. Not only the form of expression has changed, but the data is now of essentially a different type.

Change state transforms may be *symmetrical* or *polarized*. A symmetrical change state transform is one where the reverse transform is as easy to perform as the forward transform. A polarized transform is where the reverse transform is much more difficult than the forward transform, if not impossible.

Most change state transforms tend to be polarized, as in the example given above; it is far more difficult to change a bitmap to a plot file compared with the other direction (ignoring the trivial solution of plotting each bit in the bitmap as a very short vector).

Also, transforms can

- Lose information
- Preserve information
- Add information

during the process of the transform.

It is easy to see how information can be lost during a transform; this loss can be either deliberate and required, or a by-product of the transform process. Of course, once lost, it is virtually impossible to replace without human intervention.

For example, in the plotfile to bitmap transform, the plotfile might contain color information, but the bitmap may be wanted in monochrome. Typically, monochrome bitmaps use shading patterns to represent color; it may be that the bitmap, to give reasonable resolution, can only use eight different shading patterns to represent different colors, while the original file had sixteen colors.

Thus, there is both required and unwanted loss in this example. The loss of color is required, but half the possible grey scales cannot be represented, giving an undesired loss of information.

Both polarized and lossy transforms are often necessary. The important thing to do is to consider whether any reversing of the process is likely to be necessary later; if so, consideration should be given to preserving the original, changing the sequence of transforms, or even redesigning the system (for example, if it seems to be necessary to update the transformed data, and then try to reverse the process). In general, the form with the highest information content should be the master, and it is this form that should be the subject of updates; other forms should be derived from this.

For example, if it is necessary to store both GML and final form versions of a document, it is always easiest to make updates to the GML version and recreate the final form automatically from that; it is almost always easier to update a vector graphic, and recreate a bitmap from that, instead of the reverse processes.

It is easy to conceive examples where all information is preserved (typical of simple reformatting transforms, for example), but more difficult to imagine a mechanical process actually adding information.

It is in fact arguable whether information is truly added, but some transforms can give the appearance of doing so.

What they actually do is use implicit information in the input, and make it explicit in the output, or expand combinations of data items in the input by use of rules or lookup tables to add apparently new information.

Often it is necessary to "add information" in a transform. An example would be from a vector format which does not contain any concept of hierarchy or segmentation (such as a simple plot file) to a file format which demands segment information. In cases such as this it is often necessary to create dummy hierarchical levels. In the absence of any specific information, these arbitrarily assigned levels may be entirely illogical; but there is essentially no alternative option. This kind of transform can often be less than totally satisfactory, no matter how much effort is put into the logic.

A simple example is a transform which takes a formatted text file, and turns it into a tagged GML file, by making assumptions about the text in the file (blocks of text separated by blank lines are paragraphs, all capital text centered in a single line at the top of a new page followed by two blank lines is a chapter heading, and so on).

It is easy to see that a program which loses information will probably also be a polarized one; usually the information that is lost cannot be recreated automatically.

An example of an "add information" transform can be found in A.3, "Conversion from BookManager READ Copy Form to GML" on page 163.

Finally, a factor that can significantly affect the complexity of a transform program is whether or not the information needed to determine the final form is locally defined in the input, or is spread all over the input data, what we might call "context-free" or "context-sensitive." Generally speaking, context-sensitive

transforms are much more difficult to write and get right than context-free ones. A context-free transform can be as simple as a global change command using an editor program; a context-sensitive one may have to inspect and analyze the whole of the input data before processing the first output record. An example of a very simple context-sensitive transform is given in A.2, "Changing Data in Context" on page 158, which was written to handle some problems in converting from GML document languages to word processing style definitions.

15.2 Round and Round We Go ...

With products like Corel Draw and OS/2 Image Support, we have seen that we can effectively combine the mainframe and workstation worlds as far as graphics and images are concerned, and use whatever we create using any particular program in most other places.

People have long wanted to do the same with formatted text, combining the use of office systems, word processors, and formatting systems on both the workstation and mainframe.

With PSF/2 and AFP Workbench for Windows, we are well on the way to integrating host and workstation display and print of formatted output from applications that can generate AFPDS With the IBMAFP printer driver, the applications that can create AFPDS now include the majority of workstation GUI applications.

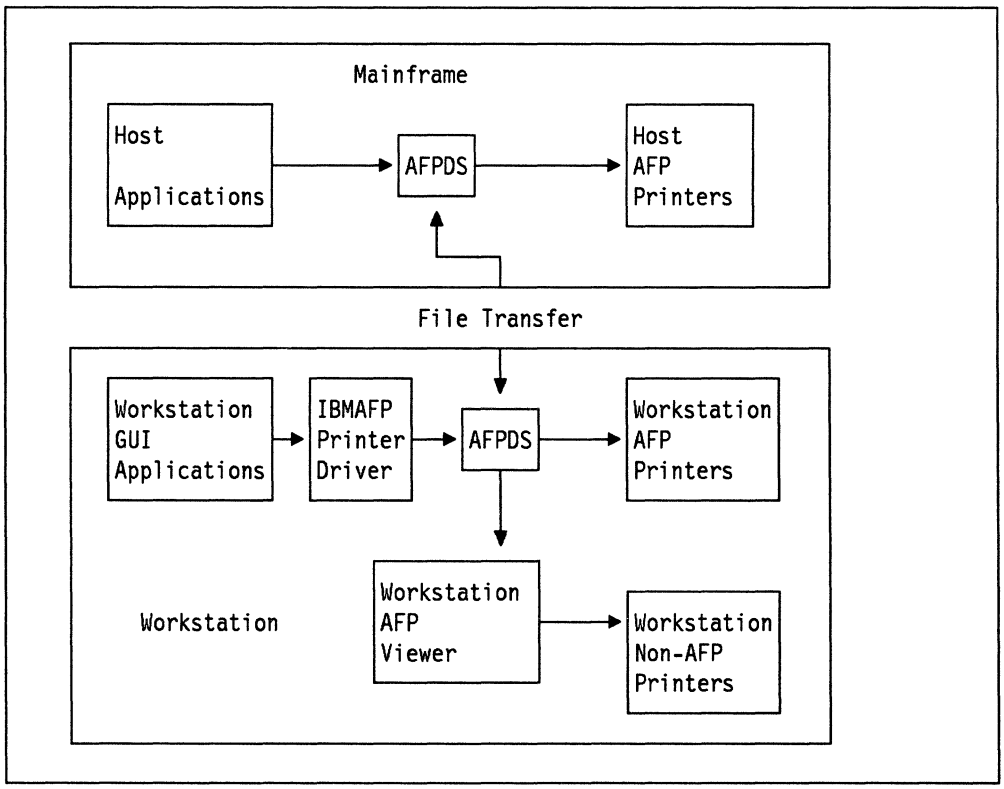


Figure 31. Integration of Host and Workstation Printing

The only thing that remains is to integrate the WYSIWYG world of word processing and desk-top with the more formal, but more powerful, capabilities of document language processing, and particularly the exciting prospects that softcopy offers for the future in the areas of textual database searching and

retrieval, multimedia integration and control, and hypertext viewing and navigation models.

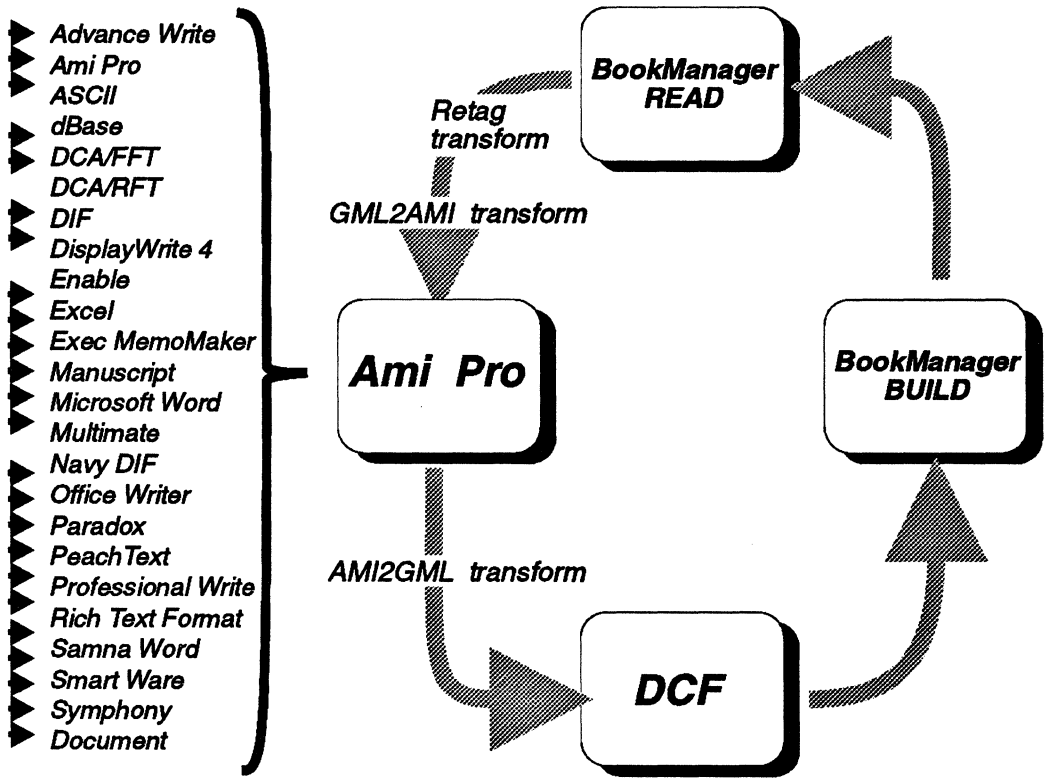


Figure 32. Text Type Transforms

The picture is still not perfect, but with products like Ami Pro, and some almost trivial filters, we can do a lot.

As an example of this, we'll show you a trivial document we sent around this loop. For the sake of starting somewhere, we created the document in Microsoft Word.

From Microsoft Word, the document was imported into Ami Pro, using the option **Keep style names**. This ensures that individual text elements retain identification.

Inside Ami Pro, a standard style sheet was overlaid on the document, and "Revert to style" for both text font and spacing selected (this overrides the values carried from Microsoft Word, and allows full style control).

The document was printed at this stage, and looks like this:

This is a sample document

This document was created in Microsoft Word as an example of text to move through various transforms.

It consists of simple text, with a top heading and a list:

1 Apples

2 Oranges

3 Pears

To see how well these simple elements can be carried from one application to another.

To check this further, we have a sub heading:

Sub-Heading

A sub-sub heading

Sub-sub heading

And one even lower:

Sub-sub-sub

The intention is to move the document through Ami Pro, use a special filter to convert exported ASCII from Ami Pro to change the document to GML, and upload it to the host for processing using DCF. There, we intend to build it as a book using BookManager BUILD.

Downloaded to the PC, BookManager READ/DOS will then be used to create an ASCII file from the softcopy book, which we will pass through a simple filter to remark the document with GML. This will then be refiltered into ASCII which Ami Pro can understand by the same filter we originally used to take Ami Pro ASCII to GML (just using another conversion profile).

At each stage, we will take a print, and see how things look. At each stage we will report how things look, and how much manual intervention was required at each stage.

Figure 33. Ami Pro Output before Upload

The file was saved both as an Ami Pro file, and an ASCII file with style names. This was put through the AMI2GML filter described in A.1, "Conversion Between Ami Pro and GML" on page 147, and the resulting GML uploaded to a VM mainframe.

The GML was processed by DCF, and the document output both on paper and through BookManager BUILD as a softcopy book. The document was visible as softcopy both on the VM host, and when downloaded back to the PC.

A standard function of both mainframe and READ/DOS is to print in untagged ASCII whatever topics of a softcopy book are requested by the reader; as well as being printed they can be copied to a file.

This was done, and the resulting ASCII file retagged to GML using the RETAG program outlined in A.3, "Conversion from BookManager READ Copy Form to GML" on page 163.

This conversion could have been done with TextTagger, but the conversion required at this level is so simple that this would have been overkill, unless the product was already available for some other purpose. See 11.4.2.1, "Converting Existing Text" on page 70 for a discussion of TextTagger versus specially written taggers.

The reverse process, from GML to word processing, though apparently simple, has one twist. Some GML tags are context dependent; how they will be formatted depends on tags around them. A simple-minded transform such as AMIGML can't handle that. In A.2, "Changing Data in Context" on page 158 we show a simple example of a context sensitive transform. This can be used as a first pass to handle some of the reverse process, such as the handling of nested list tags of different types.

When the results of using this were passed again through the AMIGML filter, this time with a profile for converting GML to Ami Pro tags, the resulting document was again loaded into Ami Pro, and the correct style sheet applied.

That process gave the following result when printed.

This is a sample document

This document was created in Microsoft Word as an example of text to move through various transforms.

It consists of simple text, with a top heading and a list:

1 Apples

2 Oranges

3 Pears

To see how well these simple elements can be carried from one application to another.

To check this further, we have a sub heading:

Sub-Heading

A sub-sub heading

Sub-sub heading

And one even lower:

Sub-sub-sub

The intention is to move the document through Ami Pro, use a special filter to convert exported ASCII from Ami Pro to change the document to GML, and upload it to the host for processing using DCF. There, we intend to build it as a book using BookManager BUILD.

Downloaded to the PC, BookManager READ/DOS will then be used to create an ASCII file from the softcopy book, which we will pass through a simple filter to remark the document with GML. This will then be refiltered into ASCII which Ami Pro can understand by the same filter we originally used to take Ami Pro ASCII to GML (just using another conversion profile).

At each stage, we will take a print, and see how things look. At each stage we will report how things look, and how much manual intervention was required at each stage.

Figure 34. Ami Pro Output after Everything

The only differences appear to be due to a minor change made in the style sheet between the two prints — the font size of the sub-sub heading — and some slight difference in text word flow, apparently due to Ami Pro respecting Microsoft Word word flow in the first instance, and following its own rules in the second.

So, it can be seen that we can, with a little ingenuity, pass documents around multiple word processors, between workstation and host formatters, and between softcopy and revisable text, and retain full information necessary to format the document appropriately.

This allows us to:

- Use a WYSIWYG workstation editor for mainframe GML
- Format and print GML on the workstation
- Archive workstation documents in softcopy using BookManager BUILD

We have done so here only for relatively simple document structures; more complex ones such as tables present more of a problem, and for example will only be partially translated from Ami Pro into GML. However, taking a pragmatic approach, either such elements are fairly rare in documentation, in which case it is acceptable to have human involvement in the transform process; or they form a large proportion of documentation, in which case it is worthwhile investing a little effort in putting more logic and capability into the simple transforms demonstrated here.

Appendix A. Source Code for Sample Transforms and EXECs

Important: The sample code in this section is presented for tutorial purposes only. While this code has been used to perform the functions stated under the system conditions experienced by the authors, the authors and IBM make no representation that this code or derivatives will necessarily work in all or any other situations, and the authors and IBM accept no responsibility for any damage or loss arising out of its use. It is entirely the responsibility of any user to ensure that this code is appropriate to their needs.

Some of the code below is known to contain imperfections and shortcomings; none is complete answer to the problems they start to address. They are offered as a proof of concept that these questions can be addressed with relatively simple tools while achieving a significant percentage of the required results.

Where-real life situations like these must be met, these examples may serve as a starting point, IBM customers may wish to use IBM services to build on them.

A.1 Conversion Between Ami Pro and GML

This section describes a simple utility to convert Ami Pro tagged ASCII files to GML and vice versa. It won't cope with all the components of files (tables will need a bit of human manipulation, for example), but it will get you a fair bit of the way there.

This lets you use programs such as Ami Pro not only to create nice printouts on the workstation, but also convert your text without rekeying into GML, so that it can be used in more structured documentation. The reverse direction is also possible; that lets you take GML files and look at them with a WYSIWYG editor, and print them nicely on the workstation also.

Note: If you want to convert from GML to Ami Pro, call the profile file for your changes "GML.PRO." If you do that, the code below will recognize that the output is required for Ami Pro, and will only put carriage returns in front of tags. This is the form required for Ami Pro to recognize paragraph sections — import it into Ami Pro with the ASCII option "carriage returns after paragraphs."

This code and documentation may be freely used inside IBM for any business-related purpose. This includes delivery of the EXEC or derivatives to customers as part of work done under a service contract, or provision to customers on an unsupported "as-is" basis. IBM will accept no responsibility for the suitability of the code for any particular purpose.

The copyright remains owned by IBM United Kingdom Limited.

A.1.1 Functional Description

The EXEC on which this code was based was originally written to convert some books written using a desk-top program to BookMaster format. As all the tags that would eventually appear in the files couldn't be defined at the start, the program was written to be table driven. This meant that it turned out not to be specific to the package for which it was originally written; it has been used to convert SGML, and tapes containing printer's codes, to BookMaster. It may have even more generality than that, but that's up to you to discover.

It was later converted into C on the PC, which is the current version.

The fact that it is table driven means it can be used with programs such as Ami Pro and Ventura Publisher, which allow the user to create their own tag names ("styles," for you Ami Pro users). If you create your own tag names, then you'll also have to modify the profile files that do the conversion, to tell the program what to change your tags to, and what to change to your tags.

It assumes that most controls will be at the start of lines. This may not always be the case. If this is so, the text needs to be split into lines beginning with the controls. This can usually be done by global change commands in an editor, or by using a multiple change filter, such as David Mitchell's excellent MULCH filter on PCTOOLS.

Controls within lines can be found, but only a simple replacement can be done for this kind of control — no clever stuff such as putting special controls before the first or after the last in a group. The most you can do with inline changes is to give a different replacement string for odd and even occurrences of the control within a line — that lets you switch highlighting on and off, for example.

The simplest kind of change is simple substitution of tag for control. Clearly the EXEC can do that. Almost as complex is the handling of blocks of controls - for example bullets or other dingbats to represent a list. Straight substitution can handle the list item tags, but this EXEC can also insert the list start and end tags.

A.1.2 Invocation

The command to invoke the program is:

```
AMIGML infilename outfilename profilefilename
```

where:

infilename Fully qualified name of input file

outfilename Fully qualified name of output file

profilefilename Fully qualified file name of change specification file.

A.1.3 Profile File

The profile file consists of List Headers which precede lists of Change Specifications. The List Header defines the type of change the Change Specifications following it will be used for. For example, the REPLACE List Header precedes the Change Specifications which define all straight substitutions.

A.1.3.1 List Headers

A List Header has the following format:

```
>>LIST identifier
```

>>LIST Identifies the line as a List Header.

identifier Identifies the type of Change Specifications which follow the header.

The Change Specification types which can be identified are:

NULL Any lines starting with control codes defined in the Change Specifications following this header are totally ignored.

REPLACE The Change Specifications following this header define simple substitutions. The control code is replaced by the substitution tag.

FOREFIRST These define substitutions which will occur only on the first occurrence of a code in a block of lines all starting with the same code. The substitution tag is placed in a separate line before the line containing the matching code. The code itself can then be subject to a match against a **REPLACE** or other substitution.

FOREALL Like **FOREFIRST** substitutions, the tags are placed in a line prior to that containing the matching code, but these occur for *all* occurrences of the matching code, not just the first in a block.

POSTALL Like **FOREALL**, but the tag is again placed on the following line.

POSTLAST Like **FORELAST**, but you get the idea.

INLINE Most Change Specifications check for control codes at the *beginning* of a line. These check for codes *anywhere* in the text. The Change Specifications for **INLINE** controls specify two tags, one for odd occurrences, and one for even occurrences.

Where only one tag is needed for a control, it is more efficient if that can be forced to the beginning of a line and handled by a REPLACE specification.

CONTINUATION (You shouldn't need to use this one with Ami Pro files, but we've left it in in case you find another use for it.) The specifications after this header define strings at the *end* of a line which are to be treated as continuation markers, which identify blocks of lines which are to be treated as a single logical line.

This program strips the continuation control sequences from the file.

A.1.3.2 Change Specifications

There are two kinds of Change Specifications, those for **INLINE** substitutions, and those for most of the others (the *Standard* Change Specification).

Standard Change Specifications: Standard Change Specifications have the following format:

```
tag << control code string
```

The *tag* is the string to be substituted for the control code. It may be all blank, in which case no tag is substituted for the matching control code.

The *control code string* is what is matched against in the input file. Where this code is found at the beginning of a line (end of a line for a **CONTINUATION** specification), it is deleted from the file and replaced by the *tag*.

A control code string may be either a character string or a hexadecimal string. A character string is defined by all characters in the line following the << marker, less all preceding and trailing blanks. All blanks within the string are significant.

A string is taken to be a hexadecimal string if it is preceded by a quotation character, and terminated by a quotation character followed by an *x*. The *x* may be either upper- or lower-case, and the quotation characters may be either the single quote or the double quote, as long as both are the same.

Inline Change Specifications: These are very similar to the standard, but they have two tags:

```
tag1 tag2 << control code string
```

Control code strings are as defined for the Standard Specification.

Tag1 is used for odd numbered occurrences of the control code matched, *tag2* for even numbered occurrences. If a code is to be replaced by the same tag in both cases, it must be specified twice. If only one tag is specified in an **INLINE** specification, then even numbered occurrences will be null substituted.

Blanks in replacement strings: If you want a blank in a string which is to replace an existing one, put a "_" character where you want the blank. This will be replaced by a blank on output.

Carriage return in replacement strings: If you want a carriage return in a string which is to replace an existing one, put a "^" character where you want the carriage return. This will be replaced by a carriage return on output.

A.1.3.3 Examples

Example 1

```
>>LIST NULL
       << /*
>>LIST INLINE
:q. :eq. << '
:q. :eq. << "
```

This simple example would produce an output file containing none of the lines beginning with /* in the input file, and with all quote marks changed to *:q. :eq.* pairs.

Example 2:

```
>>LIST FOREFIRST
:nl.   << <List>
>>LIST REPLACE
:li.   << <List>
>>LIST POSTLAST
:enl.  << <List>
```

This example would, if given this input:

```
any old text
<List> first item
<List> second item
<List> third
followed by other text
```

produce the following output:

```
any old text
:nl.
:li. first item
:li. second item
:li. third
:enl.
followed by other text
```

Example 3 — Converting an Ami Pro file: Say you have an Ami Pro file containing just:

```
<Heading>Memo
<Body Text>To Fred
<Body Text>From Harry
<Body Single>12th June
<Body Text>I hope you managed to sort out the production line. I need
<Numbered List>4 sprockets,
<Numbered List>a widget, and
<Numbered List>43 doohickeys
<Body Text> by Monday at the latest!
```

You can get a plain ASCII file out of Ami Pro, with the tags intact as above, by using the **Save As ..** menu, selecting **ASCII** as the output file type, and making sure that you select the **ASCII Options ..** "CR/LF after lines" and "Keep style names."

The sample AMI.PRO profile file —

```
>>LIST FOREFIRST
:ol.        << <Numbered List>
:ul.        << <Indent>
>>LIST POSTLAST
:eol.       << <Numbered List>
:eul.       << <Indent>
:etable.    << <Table Text>
:ethd.^:row. << <Table Heading>
>>LIST REPLACE
:li.        << <Numbered List>
:h1.        << <Heading>
:h1.        << <Title>
:h2.        << <Subhead>
:li.        << <Indent>
:p.         << <Body Text>
:row.       << <Table Text>
            << <Body Single>
>>LIST INLINE
:table.^:thd. :table.^:thd. << <Table Heading>
```

will convert this to a GML file like:

```
:h1.Memo
:p.To Fred
:p.From Harry
12th June
:p.I hope you managed to sort out the production line. I need
:ol.
:li.4 sprockets,
:li.a widget, and
:li.43 doohickeys
:eol.
:p.by Monday at the latest!
```

Not very sophisticated, perhaps, but it's up to you to extend it!

The exhausting problem of writing a profile to reverse the process is left as an exercise for the student.

A.1.3.4 Sample Code: Change AMI PRO ASCII Files to GML (and back):

```
/*
     AMIGML.C
     Change AMI PRO ASCII files with tags to GML
*/

#define MAXNL 100
#define MAXFF 50
#define MAXFA 50
#define MAXFL 50
#define MAXPF 50
#define MAXPA 50
#define MAXPL 50
#define MAXRR 150
#define MAXCT 5
#define MAXIL 50
#define REPSIZE 40
#define TAGSIZE 40

/* Requires three arguments; first is input filename,
   second is output filename,
   the third is the profile file name. */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void init(void);
int  checknull(void);
void presub(void);
void chglin(void);
void postsb(void);
void inline(char *);
void getword(char *,char *,char *);
void gettest(char *,int,char *);
/* void parse(char *, char *, char, char *); */
void substr(char *,char *,int,int);
int  gettwo(char *,char *);
int  strip(char *);
void hexit(char *);
void cchar(void);
void chknxt(void);

FILE *infile,*outfile,*chgfile;

char lastkl[REPSIZE];
char nextkl[REPSIZE];
int  savcont;
int  contchar;
int  match;
int  wasblank;
int  gml2ami;
int  list;
int  hilite;
int  indent;
int  firstblank;
int  lastsize;
char instr1[1024];
char instr2[1024];
char outstr[1024];

/* Configuration arrays */
char nlrstr[MAXNL][REPSIZE];
int  nlidx;
char ffrstr[MAXFF][REPSIZE];
char fftag[MAXFF][TAGSIZE];
int  ffflag[MAXFF];
int  ffidx;
char farstr[MAXFA][REPSIZE];
char fatag[MAXFA][TAGSIZE];
int  faidx;
char flrstr[MAXFL][REPSIZE];
char fltag[MAXFL][TAGSIZE];
int  flidx;
char pfrstr[MAXPF][REPSIZE];
char pftag[MAXPF][TAGSIZE];
char pfrstridx[MAXPF];
int  pfidx;
char parstr[MAXPA][REPSIZE];
char patag[MAXPA][TAGSIZE];
int  paidx;
char plrstr[MAXPL][REPSIZE];
char pltag[MAXPL][TAGSIZE];
int  plidx;
char rrrstr[MAXRR][REPSIZE];
char rrtag[MAXRR][TAGSIZE];
int  rridx;
char ilrstr[MAXIL][REPSIZE];
char iloddtag[MAXIL][TAGSIZE];
char ilevntag[MAXFF][TAGSIZE];
int  ilodd[MAXIL];
int  ilidx;
char ctrstr[MAXCT][REPSIZE];
int  ctidx;


main(argc, argv, envp)
   int argc;
   char *argv[];
   char *envp[];
{
   int lnum;
   int  endit=0;
   int scc,ssc;

   gml2ami=0;
   instr1[0]='\0';
   instr2[0]='\0';


   /* Check input parameters */
   if (argv[1]==NULL) {
      fprintf(stderr,"No input file specified\n");
      return 4;
   }
   if (argv[2]==NULL) {
      fprintf(stderr,"No output file specified\n");
      return 4;
   }
   if (argv[3]==NULL) {
      fprintf(stderr,"No changes file specified\n");
      return 4;
   }

   /* Open files */
   if ((infile=fopen(argv[1],"r"))==0) {
      fprintf(stderr,"Unable to open input file %s\n",argv[1]);
      return 6;
   } /* endif */
   if ((outfile=fopen(argv[2],"w"))==0) {
      fprintf(stderr,"Unable to open output file %s\n",argv[2]);
      return 6;
   } /* endif */
   if ((chgfile=fopen(argv[3],"r"))==0) {
      fprintf(stderr,"Unable to open profile file %s\n",argv[2]);
      return 6;
   } /* endif */

   /* Main Processing */
   printf("\nAMI2GML: Copyright IBM United Kingdom 1992
           All Rights Reserved\n\n");
   init();
   lnum=0;
   if (strlen(argv[3])>6)
               substr(instr1,argv[3],strlen(argv[3])-7,7);
   if (strcmpl(instr1,"GML.PRO")==0) {
     gml2ami=1;
```

```
    printf("Converting GML to Ami Pro tagged ASCII\n\n");
  }
  instr1[0]='\0';
  while (!endit) {
     endit=gettwo(instr1,instr2);
     lnum++;
     if (endit) break;
     if (checknull()) continue;
     presub();
     scc=contchar;
     ssc=savcont;
     chglin();
     contchar=scc;
     savcont=ssc;
     postsb();
     savcont=0;
     if (contchar==1) savcont=1;
  }

  fclose(infile);
  fclose(outfile);

  return;
}

int checknull()
{
  char test[REPSIZE];
  int i;

  int rc=0;
  for (i=0;i<nlidx;i++) {
    gettest(test,strlen(nlrstr[i]),instr1);
    if (strcmp(test,nlrstr[i])==NULL) {
      rc=1;
      break;
    }
  }
  return rc;
}

/* Read change intructions file and initialise */
void init()
{
  char line[1024];
  char key[40];
  char test[40];
  char parm[40];
  char itag[256];
  char rest[1024];
  char rstring[1024];
  char rstr2[1024];
  char sep[40];
  int i;

  lastk1[0]='\0';
  savcont=0;
  match=0;
  list=0;
  hilite=0;
  indent=0;
  firstblank=1;
  lastsize=100;

  nlidx=0;
  ffidx=0;
  faidx=0;
  flidx=0;
  pfidx=0;
  paidx=0;
  plidx=0;
  rridx=0;
  ilidx=0;
  ctidx=0;

fgets(line,1023,chgfile);
while (1) {
  getword(key,rest,line);
  if (strcmp(key,">>LIST")==NULL) {
    while (strcmp(key,">>LIST")==NULL) {
      if (fgets(line,1023,chgfile)==NULL) return;
      if ((line[0]=='>')&(line[1]=='>')) break;
      getword(itag,rstr2,line);
      if (strcmp(itag,"<<")==NULL) {
        itag[0]='\0';
        strcpy(rstring,rstr2);
      } else {
        getword(sep,rstring,rstr2);
      }
      strip(rstring);
      strip(itag);
      for (i=0;i<=strlen(itag);i++) {
        if (itag[i]=='_') itag[i]=' ';
        if (itag[i]=='^') itag[i]='\n';
      }
      strip(rest);
      hexit(rstring);
      if (strcmp(rest,"NULL")==NULL) {
        if (nlidx<MAXNL) {
          strcpy(nlrstr[nlidx],rstring);
          nlidx++;
        } else {
          fprintf(stderr,
            "Number of NULL identifiers exceeded (%d).\n",MAXNL);
        }
        continue;
      }
      if (strcmp(rest,"FOREFIRST")==NULL) {
        if (ffidx<MAXFF) {
          strcpy(ffrstr[ffidx],rstring);
          strcpy(fftag[ffidx],itag);
          ffflag[ffidx]=1;
          ffidx++;
        } else {
          fprintf(stderr,
            "Number of FOREFIRST identifiers exceeded (%d).\n",
                                                          MAXFF);
        }
        continue;
      }
      if (strcmp(rest,"FOREALL")==NULL) {
        if (faidx<MAXFA) {
          strcpy(farstr[faidx],rstring);
          strcpy(fatag[faidx],itag);
          faidx++;
        } else {
          fprintf(stderr,
            "Number of FOREALL identifiers exceeded (%d).\n",
                                                          MAXFA);
        }
        continue;
      }
      if (strcmp(rest,"FORELAST")==NULL) {
        if (flidx<MAXFL) {
          strcpy(flrstr[flidx],rstring);
          strcpy(fltag[flidx],itag);
          flidx++;
        } else {
          fprintf(stderr,
            "Number of FORELAST identifiers exceeded (%d).\n",
                                                          MAXFL);
        }
        continue;
      }
      if (strcmp(rest,"POSTFIRST")==NULL) {
        if (pfidx<MAXPF) {
          strcpy(pfrstr[pfidx],rstring);
          strcpy(pftag[pfidx],itag);
          pfidx++;
        } else {
```

```
    fprintf(stderr,
      "Number of POSTFIRST identifiers exceeded (%d).\n",
                                                  MAXPF);
  }
  continue;
}
if (strcmp(rest,"POSTALL")==NULL) {
  if (paidx<MAXPA) {
    strcpy(parstr[paidx],rstring);
    strcpy(patag[paidx],itag);
    paidx++;
  } else {
    fprintf(stderr,
      "Number of POSTALL identifiers exceeded (%d).\n",
                                                 MAXPA);
  }
  continue;
}
if (strcmp(rest,"POSTLAST")==NULL) {
  if (plidx<MAXPL) {
    strcpy(plrstr[plidx],rstring);
    strcpy(pltag[plidx],itag);
    plidx++;
  } else {
    fprintf(stderr,
      "Number of POSTLAST identifiers exceeded (%d).\n",
                                                 MAXPL);
  }
  continue;
}
if (strcmp(rest,"REPLACE")==NULL) {
  if (rridx<MAXRR) {
    strcpy(rrrstr[rridx],rstring);
    strcpy(rrtag[rridx],itag);
    rridx++;
  } else {
    fprintf(stderr,
      "Number of REPLACE identifiers exceeded (%d).\n",
                                                 MAXRR);
  }
  continue;
}
if (strcmp(rest,"INLINE")==NULL) {
  if (ilidx<MAXIL) {
    if (strcmp(itag,">>")==NULL) {
      iloddtag[ilidx][0]='\0';
      ilevntag[ilidx][0]='\0';
      strip(rstr2);
      strcpy(ilrstr[ilidx],rstr2);
    } else {
      strcpy(iloddtag[ilidx],itag);
      getword(ilevntag[ilidx],rstring,rstr2);
      strip(ilevntag[ilidx]);
      for (i=0;i<=strlen(ilevntag[ilidx]);i++) {
        if (ilevntag[ilidx][i]=='_')
                ilevntag[ilidx][i]=' ';
        if (ilevntag[ilidx][i]=='^')
                ilevntag[ilidx][i]='\n';
      }
      getword(sep,rstr2,rstring);
      strip(rstr2);
      strcpy(ilrstr[ilidx],rstr2);
      ilidx++;
    }
  } else {
    fprintf(stderr,
      "Number of INLINE identifiers exceeded (%d).\n",
                                                 MAXIL);
  }
  continue;
}
if (strcmp(rest,"CONTINUATION")==NULL) {
  if (ctidx<MAXCT) {
    strcpy(ctrstr[ctidx],rstring);
    ctidx++;
  } else {
    fprintf(stderr,
      "Number of CONTINUATION identifiers exceeded (%d).\n"
                                                 ,MAXCT);
  }
  continue;
 }
 }
} else {
  fprintf(stderr,
    "Changes file does not contain valid specifications.\n);
  fclose(infile);
  fclose(outfile);
  fclose(chgfile);
  exit(16);
 }
}

 return;
}

/* Snarf first word from input line. */
void getword(word,rest,line)
char *word;
char *rest;
char *line;
{
 int i,j,flag;

 j=0;
 flag=0;
 word[0]='\0';
 rest[0]='\0';

 for (i=0;i<=strlen(line);i++) {
  if (!flag) {
   if (line[i]==' ') continue;
   else flag=1;
  }
  if ((line[i]==' ')&(flag==1)) {
   word[j]='\0';
   j=0;
   flag=2;
  }
  if (flag==1) word[j++]=line[i];
  else rest[j++]=line[i];
 }
 rest[j]='\0';
 strip(rest);
 if (rest[strlen(rest)-1]=='\n')
                 rest[strlen(rest)-1]='\0';
 return;
}


/* Get comparison string from input line. */
void gettest(compstr,clen,line)
char *compstr;
int clen;
char *line;
{
 int i;

 for (i=0;i<clen;i++) {
  compstr[i]=line[i];
 }
 compstr[i]='\0';
 return;
}


/* Convert a string in valid REXX hex notation to a string */
void hexit(string)
char *string;
{
```

```
  char c1,c2;
  char digit[3];
  int last;
  int temp;
  int i;
  last=strlen(string)-1;

  if ((string[last]=='x')||(string[last]=='X')) {
    c1=string[0];
    c2=string[last-1];
    if (((c1=='\'')&(c2=='\''))||((c1=='"')&(c2=='"')))
{
      digit[1]='\0';
      for (i=0;i<((last-1)/2);i++) {
        digit[0]=string[i+i+1];
        if (digit[0]=='A') strcpy(digit,"10");
        else if (digit[0]=='B') strcpy(digit,"11");
        else if (digit[0]=='C') strcpy(digit,"12");
        else if (digit[0]=='D') strcpy(digit,"13");
        else if (digit[0]=='E') strcpy(digit,"14");
        else if (digit[0]=='F') strcpy(digit,"15");
        else if (digit[0]=='a') strcpy(digit,"10");
        else if (digit[0]=='b') strcpy(digit,"11");
        else if (digit[0]=='c') strcpy(digit,"12");
        else if (digit[0]=='d') strcpy(digit,"13");
        else if (digit[0]=='e') strcpy(digit,"14");
        else if (digit[0]=='f') strcpy(digit,"15");
        temp=atoi(digit)*16;
        digit[0]=string[i+i+2];
        digit[1]='\0';
        if (digit[0]=='A') strcpy(digit,"10");
        else if (digit[0]=='B') strcpy(digit,"11");
        else if (digit[0]=='C') strcpy(digit,"12");
        else if (digit[0]=='D') strcpy(digit,"13");
        else if (digit[0]=='E') strcpy(digit,"14");
        else if (digit[0]=='F') strcpy(digit,"15");
        else if (digit[0]=='a') strcpy(digit,"10");
        else if (digit[0]=='b') strcpy(digit,"11");
        else if (digit[0]=='c') strcpy(digit,"12");
        else if (digit[0]=='d') strcpy(digit,"13");
        else if (digit[0]=='e') strcpy(digit,"14");
        else if (digit[0]=='f') strcpy(digit,"15");
        temp=temp+atoi(digit);
        string[i]=(char)temp;
        string[i+1]='\0';
      }
    }
  }
  return;
}


void parse(line,pref,sepa,rest)
char *line;
char *pref;
char sepa;
char *rest;
{
  int i,j;
  int test=0;
  j=0;

  pref[0]='\0'; /* Strings null before start. */
  rest[0]='\0';

  for (i=0;i<=strlen(line);i++) {
    if (line[i]==sepa) {
      test=1;
      pref[i]='\0';
      continue;
    }
    if (!test) {
      pref[i]=line[i];
    } else {
      rest[j++]=line[i];
    }
  }
  rest[j]='\0';
  return;
}


/* Strip leading and trailing blanks from string */
int strip(string)
char *string;
{
  int i,end;
  int start=-1;
  int indent=0;
  end=strlen(string);

  for (i=0;i<strlen(string);i++) {
    if (string[i]!=' ') break;
    indent++;
    start=i;
  }
  for (i=(strlen(string)-1);i>0;i--) {
    if (string[i]=='\n') continue;
    if (string[i]!=' ') break;
    end=i;
  }
  if ((start!=-1)||(end!=strlen(string))) {
    for (i=0;i<(end-start-1);i++) {
      string[i]=string[i+start+1];
    }
    string[i]='\0';
  }
  return indent;
}

void postsb()
{
  char test[REPSIZE];
  int i;

  chknxt();

  for (i=0;i<paidx;i++) {
    gettest(test,strlen(parstr[i]),instr1);
    if (strcmp(parstr[i],test)==NULL) {
      if (!contchar) {
        strcpy(outstr,patag[i]);
        if (gml2ami) {
          fputc('\n',outfile);
          strcat(outstr," ");
        } else {
          strcat(outstr,"\n");
        }
        fputs(outstr,outfile);
        strcpy(lastkl,test);
        match=1;
        break;
      }
    } else if ((strcmp(parstr[i],lastkl)==NULL)&(!contchar)&
                                          (savcont)) {
      strcpy(outstr,patag[i]);
      if (gml2ami) {
        fputc('\n',outfile);
        strcat(outstr," ");
      } else {
        strcat(outstr,"\n");
      }
      fputs(outstr,outfile);
      strcpy(lastkl,test);
      match=1;
      break;
    }
  }

  for (i=0;i<pfidx;i++) {
```

```
    gettest(test,strlen(pfrstr[i]),instr1);
    if ((strcmp(pfrstr[i],test)==NULL)&(pfrstridx[i])) {
      if (!contchar) {
        strcpy(outstr,pftag[i]);
        if (gml2ami) {
          fputc('\n',outfile);
          strcat(outstr," ");
        } else {
          strcat(outstr,"\n");
        }
        fputs(outstr,outfile);
        strcpy(lastkl,test);
        match=1;
        break;
      }
    } else if ((strcmp(pfrstr[i],lastkl)==NULL)&(!contchar)&
                                (savcont)) {
      strcpy(outstr,pftag[i]);
      if (gml2ami) {
        fputc('\n',outfile);
        strcat(outstr," ");
      } else {
        strcat(outstr,"\n");
      }
      fputs(outstr,outfile);
      strcpy(lastkl,test);
      match=1;
      break;
    } else if (strcmp(pfrstr[i],test)==NULL) {
      pfrstridx[i]=1;
    }
  }

  for (i=0;i<plidx;i++) {
    gettest(test,strlen(plrstr[i]),instr1);
    substr(nextkl,instr2,0,strlen(plrstr[i]));
    if (strcmp(plrstr[i],test)==NULL) {
      if ((strcmp(plrstr[i],nextkl)!=NULL)&(!contchar)) {
        strcpy(outstr,pltag[i]);
        if (gml2ami) {
          fputc('\n',outfile);
          strcat(outstr," ");
        } else {
          strcat(outstr,"\n");
        }
        fputs(outstr,outfile);
        strcpy(lastkl,test);
        match=1;
        break;
      }
    }
    if ((strcmp(plrstr[i],lastkl)==NULL)&
          (strcmp(plrstr[i],nextkl)!=NULL)&
              (!contchar)&(savcont)) {
      strcpy(outstr,pltag[i]);
      if (gml2ami) {
        fputc('\n',outfile);
        strcat(outstr," ");
      } else {
        strcat(outstr,"\n");
      }
      fputs(outstr,outfile);
      strcpy(lastkl,test);
      match=1;
      break;
    }
  }
/*  if ((!match)&(!savcont)&(!contchar)) {
    lastkl[0]='\0';
  }
  if (contchar) lastsize=100;
*/

  return;
}

void presub()
{
  char test[REPSIZE];
  int i;

  chknxt();

  for (i=0;i<faidx;i++) {
    gettest(test,strlen(farstr[i]),instr1);
    if (strcmp(farstr[i],test)==NULL) {
      strcpy(outstr,fatag[i]);
      if (gml2ami) {
        fputc('\n',outfile);
        strcat(outstr," ");
      } else {
        strcat(outstr,"\n");
      }
      fputs(outstr,outfile);
      strcpy(lastkl,test);
      match=1;
      break;
    }
  }

  for (i=0;i<ffidx;i++) {
    gettest(test,strlen(ffrstr[i]),instr1);
    if ((strcmp(ffrstr[i],test)==NULL)&(fffIag[i])&(
strcmp(lastkl,ffrstr[i])!=NULL)) {
      strcpy(outstr,fftag[i]);
      if (gml2ami) {
        fputc('\n',outfile);
        strcat(outstr," ");
      } else {
        strcat(outstr,"\n");
      }
      fputs(outstr,outfile);
      strcpy(lastkl,test);
      match=1;
      fffIag[i]=0;
      break;
    } else if (strcmp(test,ffrstr[i])==NULL) {
      fffIag[i]=1;
    }
  }

  for (i=0;i<flidx;i++) {
    gettest(test,strlen(flrstr[i]),instr2);
    if (strcmp(flrstr[i],test)==NULL) {
      strcpy(outstr,fltag[i]);
      if (gml2ami) {
        fputc('\n',outfile);
        strcat(outstr," ");
      } else {
        strcat(outstr,"\n");
      }
      fputs(outstr,outfile);
      strcpy(lastkl,test);
      match=1;
      break;
    }
  }

  return;
}

void chglin()
{
  int i;
  int ltst;
  int wrote;
  char kl[REPSIZE];
  char temp[1024];

  savcont=contchar;
```

```
  cchar();
  inline(instr1);
  wrote=0;
  for (i=0;i<rridx;i++) {
    ltst=strlen(rrrstr[i]);
    gettest(k1,ltst,instr1);
    if (strcmp(k1,rrrstr[i])==NULL) {
      substr(temp,instr1,ltst,strlen(instr1)-ltst);
      strip(temp);
      strcpy(outstr,rrtag[i]);
      strcat(outstr,temp);
      if (gml2ami) {
        outstr[strlen(outstr)-1]='\0';
        fputc('\n',outfile);
      }
      fprintf(outfile,outstr);
      if (gml2ami) fputc(' ',outfile);
      strcpy(lastk1,k1);
      match=1;
      wrote=1;
      lastsize=100;
      break;
    } else if ((strcmp(rrrstr[i],lastk1)==NULL)&
                          (strcmp(rrrstr[i],k1)==NULL)) {
      strcpy(lastk1,k1);
      wrote=0;
      break;
    }
  }
  if (!wrote) {
    strcpy(outstr,instr1);
    ltst=strlen(outstr);
    if (!ltst) ltst=1;
    if (!lastsize) lastsize=1;
    if (gml2ami) {
      outstr[strlen(outstr)-1]='\0';
    }
    fprintf(outfile,outstr);
    if (gml2ami) fputc(' ',outfile);
    lastsize=ltst;
    match=0;
  }

  return;
}

void cchar()
{
  int i,j,k;
  int testlen;
  char test[1024];
  char k1[REPSIZE];

  contchar=0;
  for (i=0;i<ctidx;i++) {
    strcpy(test,instr1);
    strip(test);
    testlen=strlen(ctrstr[i]);
    if (testlen==0) return;
    for (j=strlen(test)-1,k=testlen-1;k>=0;j--,k--) {
      k1[k]=test[j];
      if (k1[k]=='\n') {
        k++;
      }
    }
    k1[testlen]='\0';
    if (strcmp(k1,ctrstr[i])==NULL) {
      contchar=1;
      break;
    }
  }
  if (contchar) {
    for (i=0;i<(strlen(test)-testlen-1);i++) {
      if (test[i]=='\0') instr1[i]=' ';
      else instr1[i]=test[i];
    }
    instr1[i]='\0';
    strip(instr1);
    i=strlen(instr1);
    if (instr1[i-1]!='\n') instr1[i++]='\n';
    instr1[i]='\0';
  }

  return;
}


/* Substring routine */
void substr(target,source,start,length)
char *target;
char *source;
int start;
int length;
{
  int i;

  for (i=0;i<=start;i++) {
    if (source[i]=='\0') {
      target[0]='\0';
      return;
    }
  }
  for (i=0;i<length;i++) {
    target[i]=source[i+start];
  }
  target[length]='\0';
  return;
}


/* Get next two non-blank lines.
            Return end=Y if eof on first line. */
int gettwo(in1,in2)
char *in1;
char *in2;
{
  char *rc;

  if (in1[0]=='\0') {
    /* Must be start of program */
    while ((in2[0]=='\n')||(in2[0]=='\0')) {
      rc=fgets(in2,1023,infile);
      if (rc==NULL) {
        return 1;
      }
    }
  }
  if (in2[0]=='\0') {
    /* Can only have been from an EOF last time. */
    return 1;
  }
  strcpy(in1,in2);
  in2[0]='\0';
  wasblank=0;
  while ((in2[0]=='\n')||(in2[0]=='\0')) {
    rc=fgets(in2,1023,infile);
    strip(in2);
    if (in2[strlen(in2)-1]!='\n') {
      strcat(in2,"\n");
    }
    if ((in2[0]=='\0')||(in2[0]=='\n')) wasblank=1;
    if (rc==NULL) {
      in2[0]='\0';
      break;
    }
  }
  return 0;
}
```

```
void inline(instr)
char *instr;
{
  int i,idx,ltst;
  char kl[REPSIZE];
  char temp[1024];

  strcpy(outstr,instr);
  for (idx=0;idx<ilidx;idx++) {
    ltst=strlen(ilrstr[idx]);
    if (strlen(instr)<ltst) continue;
    for (i=0;i<=(strlen(instr)-ltst);i++) {
      substr(kl,instr,i,ltst);
      if (strcmp(kl,ilrstr[idx])==NULL) {
        if (ilodd[idx]) ilodd[idx]=0;
        else ilodd[idx]=1;
        if (i>1) substr(outstr,instr,0,i);
        else outstr[0]='\0';
        if (ilodd[idx]) strcat(outstr,iloddtag[idx]);
        else strcat(outstr,ilevntag[idx]);
        if ((i+ltst-1)<strlen(instr)) {
          substr(temp,instr,i+ltst,(strlen(instr)-i-ltst));
          strcat(outstr,temp);
        }
        strcpy(instr,outstr);
      }
    }
  }
  return;
}

void chknxt()
{
  char test[REPSIZE];
  int i;

  /* We assume that lines with no tag are continuations
     of the last tag; Ami Pro separates elements with a
     blank line.
     You may wish to null this routine for programs
     which do not do this, but use continuation characters
     (such as Ventura). */

  /* Return with contchar=0 if next line is blank or a tag,
               otherwise 1 */


  if (strlen(instr2)==0) {
    contchar=0;
    return;
  }

  contchar=1;
  for (i=0;i<nlidx;i++) {
    gettest(test,strlen(nlrstr[i]),instr2);
    if (strcmp(nlrstr[i],test)==NULL) {
      contchar=0;
      return;
    }
  }
  for (i=0;i<ffidx;i++) {
    gettest(test,strlen(ffrstr[i]),instr2);
    if (strcmp(ffrstr[i],test)==NULL) {
      contchar=0;
      return;
    }
  }
  for (i=0;i<faidx;i++) {
    gettest(test,strlen(farstr[i]),instr2);
    if (strcmp(farstr[i],test)==NULL) {
      contchar=0;
      return;
    }
  }
  for (i=0;i<flidx;i++) {
    gettest(test,strlen(flrstr[i]),instr2);
    if (strcmp(flrstr[i],test)==NULL) {
      contchar=0;
      return;
    }
  }
  for (i=0;i<pfidx;i++) {
    gettest(test,strlen(pfrstr[i]),instr2);
    if (strcmp(pfrstr[i],test)==NULL) {
      contchar=0;
      return;
    }
  }
  for (i=0;i<paidx;i++) {
    gettest(test,strlen(parstr[i]),instr2);
    if (strcmp(parstr[i],test)==NULL) {
      contchar=0;
      return;
    }
  }
  for (i=0;i<plidx;i++) {
    gettest(test,strlen(plrstr[i]),instr2);
    if (strcmp(plrstr[i],test)==NULL) {
      contchar=0;
      return;
    }
  }
  for (i=0;i<rridx;i++) {
    gettest(test,strlen(rrrstr[i]),instr2);
    if (strcmp(rrrstr[i],test)==NULL) {
      contchar=0;
      return;
    }
  }
  for (i=0;i<ilidx;i++) {
    gettest(test,strlen(ilrstr[i]),instr2);
    if (strcmp(ilrstr[i],test)==NULL) {
      contchar=0;
      return;
    }
  }

  return;
}
```

A.2 Changing Data in Context

One problem that often arises when changing data from one form to another is that there is not always a one-for-one correspondence in the changes we want to make. A does not always have to become B; it sometimes has to become C, depending on context.

We have exactly this problem with converting GML files to word processor files with style controls. The classic example is with lists, though there are others as well.

In GML, we tag all list items as such - we depend on what list is currently active to determine what kind of an item it is - what we might call the context. So, in te following:

```
:ol.
:li.
:li.
:ul.
:li.
:li.
:eul.
:li.
:eol.
```

the first two items are part of an ordered list, the second two are part of a bulleted list, and the final one is part of the previous ordered list (this "nesting" being a particularly powerful feature of document languages in general).

But it does give us the problem that we can't do a simple global change on the :li.s to some style control, or even use a filter like the previous AMIGML program.

We need a program that will recognize context, and there's a simple example of one below. It's restricted in that it only changes things (tags or whatever) that are at the front of input lines, but that's fine for the majority of GML. If you need something that handles inline strings, it's an interesting exercise for you to extend it. We've made it table driven, so you can change the way it works without having to rewrite the code. With it and AMIGML, you should be able to do quite a respectable job of converting GML into, say, Ami Pro, though being table driven, you can probably make it useful for getting to your favorite word processor or DTP program, though you'll almost certainly have to use it in conjunction with other change filters as well.

Each line in the profile file must have six entries (as is usual in such sample code, the error checking is totally absent). The entries are, in order:

1. The group number of the context
2. An indicator to say whether this context increases indentation
3. The context indicator string
4. The context closing string
5. The string that we want to replace
6. The string we want to replace it by in this context

Sounds horrid, but it's actually quite simple.

Ignoring the group number and the indentation code for the moment, take the profile line;

```
1 y   :ol.   :eol.   :li.   <nlist>
```

This says that in the input file, wherever there is a string ":li." between an ":ol." and an ":eol.," we want it replaced by the string "<nlist>"

The strings mustn't have any blanks or carriage returns in them, by the way. If you want any of the strings to have these for any reason, use an _ character for a blank, and a ^ for a carriage return.

The group number helps us with nesting. If we have:

```
1 y   :ol.   :eol.   :li.   <nlist>
1 y   :ul.   :eul.   :li.   <ulist>
```

it says that only one of the contexts can be effective at the same time, and tells the program to remember the previous states in the group, and the sequence in which they were switched on, so that when they're switched off in turn, it can tell which to carry on using.

The indentation codes are quite simple; a "y" says that the context increases indentation. So the first one above says that an :ol. tag increases indentation, and an :eol. tag decreases it. A code of "" says that this tag doesn't affect indentation one way or the other. Finally, a code of "r" says that this tag resets indentation to 0. So for example, we'd want to give this value to major header tags. Since these probably don't do much else to the context as far as we are concerned, when we have an entry for headers like this, we just use the same header tag as the end of context indicator as well; and we might just as well change them to themselves while we're at it (that's just a way of making sure that the profile entry doesn't do much other than just reset the indentation — we've shown a couple of examples below).

Contexts/replacement string pairs that are independent of each other we give different group numbers, so they don't interfere with each other. So, we might have:

```
1 y :ol.  :eol.  :li.   :lin.
1 y :ul.  :eul.  :li.   :liu.
1 y :sl.  :esl.  :li.   :lis.
2 n :fig. :efig. :xmp.  :cgr.
3 n :fig. :efig. :exmp. :ecgr.
4 r :h1.  :h1.   :h2.   :h1.
5 r :h2.  :h2.   :h2.   :h2.
```

as a simple profile to help convert from GML to Ami Pro. After passing through the CONTEXT filter, then the list items in the file will be uniquely identified so that the AMIGML program can now be used to convert it fully into a form Ami Pro can read, with all the elements uniquely identified so they can be assigned different styles.

In this case, we'll have changed :xmp.s inside figures to something else as well. Note that as far as we're concerned, when we're changing them, the start and end tags have *independent* group numbers. The same context on/context off string pairs can appear as many times as we need, as with the figure ones above; each line only determines the context for one change specification. The lines in the profile can be in any order.

When you use this filter to signify indentation like this, what you get out are the replacement strings you specify, plus a number *after the first character* defining the indentation level. With this simple example, what happens if the indentation level goes above 9 is undefined — almost certainly something nasty.

Of course, you'd probably set it all up with a simple batch command file so that you don't have to remember all the commands and profile names. The way to invoke this filter on its own is:

```
CONTEXT inputfilename outputfilename profilefilename
```

Here's the source code for the filter itself.

A.2.1 Sample Code: Simple Context-Sensitive Filter

```
/*
CONTEXT.C
    Simple table-driven context-sensitive
      filter program
*/

#include <stdio.h>
#include <string.h>

/* MAXPRO is maximum number of profile entries */
#define  MAXPRO 100
/* MAXLEN is maximum length of context, tag,
                      and replacement strings */
#define  MAXLEN 20

void   inpro(void);
void   substr(char *,char *,int,int);
void   getword(char *,char *,char *);
int    strip(char *);

char  context[MAXPRO][MAXLEN];
char  conoff[MAXPRO][MAXLEN];
char  tag[MAXPRO][MAXLEN];
char  rep[MAXPRO][MAXLEN];
char  instr[1024];
char  rest[1024];
char  outstr[1024];

int   npro;
int   cgroup[MAXPRO];
int   iscon[MAXPRO];
char  indind[MAXPRO];
int   level;

FILE *infile,*outfile,*profile;

main(argc, argv, envp)
   int argc;
   char *argv[];
   char *envp[];
{
   int i,l1,l2,j,thistime;
   long linum;
   char test[20];

   /* Check input parameters */
   if (argv[1]==NULL) {
      fprintf(stderr,"No input file specified\n");
      return 4;
   }
   if (argv[2]==NULL) {
      fprintf(stderr,"No output file specified\n");
      return 4;
   }
   if (argv[3]==NULL) {
      fprintf(stderr,"No profile file specified\n");
      return 4;
   }

   /* Open files */
   if ((infile=fopen(argv[1],"r"))==0) {
      fprintf(stderr,"Unable to open input file %s\n",argv[1]);
      return 6;
   } /* endif */
   if ((outfile=fopen(argv[2],"w"))==0) {
      fprintf(stderr,"Unable to open output file %s\n",argv[2]);
      return 6;
   } /* endif */
   if ((profile=fopen(argv[3],"r"))==0) {
      fprintf(stderr,"Unable to open profile file %s\n",argv[3]);
      return 6;
   } /* endif */

   /* Main Processing */
   linum=0;
   inpro();
   if (npro==0) {
      fprintf(stderr,"Profile file %s empty\n",argv[3]);
      return 8;
   }
   for (i=0;i<MAXPRO;i++) {
     iscon[i]=0;
   }
   while (fgets(instr,1023,infile)!=NULL) {
      linum++;
      /* For each record, check if it is switching a
                                                  context on */
      for (i=0;i<npro;i++) {
        l1=strlen(context[i]);
        strncpy(test,instr,l1);
        test[l1]='\0';
        if (strcmp(test,context[i])==0) {
          iscon[i]=1;
          /* A context switched on flips any in same group */
          for (j=0;j<npro;j++) {
            if (iscon[j]==0) continue;
            if (j==i) continue;
            if (cgroup[j]==cgroup[i]) {
              if (iscon[j]==1) iscon[j]=-1;
              else iscon[j]=iscon[j]-1;
            }
          }
          /* Check if the context is increasing indentation */
          if (indind[i]=='y') level++;
          if (indind[i]=='r') level=0;
          continue;
        }
      }
      /* Or off */
      thistime=0;
      for (i=0;i<npro;i++) {
        l1=strlen(conoff[i]);
```

```
        strncpy(test,instr,l1);
        test[l1]='\0';
        if (strcmp(test,conoff[i])==0) {
          iscon[i]=0;
          /* A context switched off flips any in same group */
          for (j=0;j<npro;j++) {
            if (iscon[j]==0) continue;
            if (j==i) continue;
            if (cgroup[j]==cgroup[i]) {
              if (iscon[j]==-1) iscon[j]=1;
              else iscon[j]=iscon[j]+1;
            }
          }
          /* Check if the context is decreasing indentation */
          if ((indind[i]=='y')&(!thistime)) {
             thistime=1;
             level--;
          }
          continue;
        }
      }
      /* Then check if it should be changed. */
      strcpy(outstr,instr);
      for (i=0;i<npro;i++) {
        /* No point checking if context off  or flipped */
        if (iscon[i]<=0) continue;
        l1=strlen(tag[i]);
        strncpy(test,instr,l1);
        test[l1]='\0';
        if (strcmp(test,tag[i])==0) {
          /* We have something to change! */
          strcpy(outstr,rep[i]);
          for (l2=l1;l2<=strlen(instr);l2++) {
            outstr[++l1]=instr[l2];
          }
          if (outstr[strlen(outstr)-1]!='\n')
             strcat(outstr,"\n");
          break;
        }
      }
      /* Finally, write the record. */
      /* If the record starts with a tag character, insert a
         level number if greater than 0 */
      if (((outstr[0]==':'||(outstr[0]=='<'))&(level>0)) {
         strcpy(instr,outstr);
         outstr[1]=(char)(level+48);
         for (i=1;i<=strlen(instr);i++) {
            outstr[i+1]=instr&lbrki&rbrk;
         }
      }
      fputs(outstr,outfile);
    } /* endwhile */

    /* Be tidy */
    fclose(infile);
    fclose(outfile);

}

/* Read profile */
void inpro(void)
{
  int i;

  npro=0;
  while (!feof(profile)) {
    /* Each input record should contain 1 numeric, a single
       character, and 4 strings */
    fscanf(profile,"%d %c %s %s %s %s",&cgroup[npro],
          &indind[npro],
          context[npro]conoff[npro],
          tag[npro],rep[npro]);
    /* cgroup is context group parameters belong to, */
    /* indind is indent indicator;
                         y=increase indentation level,
                         n=don't,
                         r=reset it to 0 */
    /* context is the string that switches context on */
    /* conoff is the string that switches it off */
    /* while on, tag */
    /* should be replaced by rep */

    /* If you want a blank in a string, use a _;
                              a CRLF, use a ^  */

    for (i=0;i<MAXLEN;i++) {
      if (context[npro][i]=='_')
                              context[npro][i]=' ';
      if (tag[npro][i]=='_')
                              tag[npro][i]=' ';
      if (rep[npro][i]=='_')
                              rep[npro][i]=' ';
      if (context[npro][i]=='^')
                             context[npro][i]='\n'
      if (tag[npro][i]=='^')
                              tag[npro][i]='\n';
      if (rep[npro][i]=='^')
                              rep[npro][i]='\n';
    }

    npro++;
  }
  npro--;
  return;
}

/* Substring routine */
void substr(char *target,char *source,int start,int length)
{
  int i;

  for (i=0;i<length;i++) {
    target[i]=source[i+start];
  }
  target[length]='\0';
  return;
}

/* Snarf first word from input line. */
void getword(char *word,char *rest,char *line)
{
  int i,j,flag;

  j=0;
  flag=0;
  word[0]='\0';
  rest[0]='\0';

  for (i=0;i<=strlen(line);i++) {
    if (!flag) {
      if (line[i]==' ') continue;
      else flag=1;
    }
    if ((line[i]==' ')&(flag==1)) {
      word[j]='\0';
      j=0;
      flag=2;
    }
    if (flag==1) word[j++]=line[i];
    else rest[j++]=line[i];
  }
  rest[j]='\0';
  strip(rest);
  return;
}

/* Strip leading and trailing blanks from string */
int strip(char *string)
{
  int i,end;
```

```
  int start=-1;
  int indent=0;
  end=strlen(string);

  for (i=0;i<strlen(string);i++) {
    if (string[i]!=' ') break;
    indent++;
    start=i;
  }
  for (i=(strlen(string)-1);i>=0;i--) {
    if (!((string[i]==' ')||
                                    (string[i]=='\n'))) break;
    end=i;
  }
  if ((start!=-1)||(end!=strlen(string))) {
    for (i=0;i<=(end-start-2);i++) {
      string[i]=string[i+start+1];
    }
    string[i]='\0';
  }
  return indent;
}
```

A.3 Conversion from BookManager READ Copy Form to GML

BookManager READ on the workstation cannot copy or print topics or ranges of topics in GML form. (The mainframe version can, though it does go overboard in its use of &rbl.s.)

This sample program is offered as both an example of a filter which can "add information," and a means of getting the output of workstation READ into GML (or mainframe READ into sensible GML). Of course it doesn't magically "add" information; it tries to deduce content from form, and it is sometimes successful.

Don't expect it to be perfect. Always validate the output from transform programs that try to reverse entropy (go the wrong way through a polarized process).

A.3.1 Sample Code: GML from BookManager READ

```
/*
   RETAG.C
   Retag BookManager copy files

   Requires two arguments; first is input filename,
                           second is output filename.
   A third parameter is optional.  If given, it specifies
   a list of dingbats by which to recognize items in an
   unordered list.
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void   init(void);
int    gettwo(char *,char *);
void   getword(char *,char *,char *);
int    strip(char *);
char   chkhd(char *);
void   substr(char *,char *,int,int);
int    chkdef(char *,char *);
int    revex(char *,char *);
void   chkpref(char *,int);
int    verify(char *,char *);

FILE   *infile,*outfile;

int    thisindent,nextindent,lastindent;
int    baseindent,itemindent;
char   outstr[1024];
char   deft[256];
char   def[256];
int    deflist;
int    lenterm;
int    termind;
int    defind;
int    revised;
int    isexamp;
int    blankbef,blankbet;
int    lineno;
int    thislen,lastlen;
int    llc;
int    thisline,nextline;
int    thisnlist,nextnlist;
char   dinglist[24];
int    contline;

char  type[40];
int   posllc[40];

main(argc, argv, envp)
   int argc;
   char *argv[];
   char *envp[];
{
   char instr1[1024];
   char instr2[1024];
   char rest[1024];
   char pref[256];
   char nprf[256];
   char nrst[1024];
   char ding[40];
   char dingline[1024];
   int  endit=0;
   int  tag;
   int  rc;
   int  xmpind;
   int  i;

   instr1[0]='\0';
   instr2[0]='\0';
   thislen=0;
   lastlen=0;
   isexamp=0;
   lineno=0;
   llc=0;
   contline=0;

   /* Check input parameters */
```

```
   if (argv[1]==NULL) {
      fprintf(stderr,"No input file specified\n");
      return 4;
   }
   if (argv[2]==NULL) {
      fprintf(stderr,"No output file specified\n");
      return 4;
   }
   if (argv[3]==NULL) {
      strcpy(dinglist," *o+-");
   } else {
      strcpy(dinglist,argv[3]);
   }


   /* Open files */
   if ((infile=fopen(argv[1],"r"))==0) {
      fprintf(stderr,"Unable to open input file %s\n",argv[1]);
      return 6;
   } /* endif */
   if ((outfile=fopen(argv[2],"w"))==0) {
      fprintf(stderr,"Unable to open output file %s\n",argv[2]);
      return 6;
   } /* endif */

   /* Main Processing */
   printf
("\n\nIBM BookManager Copy file to GML filter sample code.\n");
   printf
("\n\nCopyright (c) 1992 IBM United Kingdom\n");
   printf
("This program remains copyright IBM United Kingdom but may be\n");
   printf
("freely used by IBM customers for any purpose provided only\n");
   printf
("that the copyright notice remain intact in both source and\n");
   printf
("executable  versions. \n\n");
   init();
   while (!endit) {
      /* Get next two text lines */
      lineno++;
      endit=gettwo(instr1,instr2);
      if (endit) break;

      /* Print and return immediately if an example */
      if (isexamp==2) {
        /* If indent decreases, finish example */
        if ((nextindent<xmpind)) {
          fputs(instr1,outfile);
          fputs(":exmp.\n",outfile);
          isexamp=0;
        } else {
          fputs(instr1,outfile);
        }
        continue;
      }
      /* Revisions, braced examples, and character graphics. */
      rc=revex(instr1,instr2);
      if (rc==1) continue;

      /* Is it a header? */
      if (chkhd(instr1)=='Y') {
        thislen=0;
        continue;
      }
      /* Is it a definition list ? */
      /* note this also copies instr1 to outstr, and strips it. */
      rc=chkdef(instr1,instr2);
      if (rc==1) continue;

      /* Start by assuming we don't tag this line */
      tag=0;

      /* If a blank before an indent increases, start example */
      if ((blankbef>0)&(thisindent>lastindent)) {
        xmpind=thisindent;
        fputs(":xmp.\n",outfile);
        for (i=0;i<thisindent;i++) fputc(' ',outfile);
        isexamp=2;
        fputs(instr1,outfile);
        /* But if it's only a single line example ... */
        if (nextindent<xmpind) {
          isexamp=0;
          fputs(":exmp.\n",outfile);
        }
        continue;
      }

      /* Now check for numeric lists */
      getword(pref,rest,instr1);
      getword(nprf,nrst,instr2);
      chkpref(pref,1);
      if ((thisline)&(!thisnlist)) {
        getword(ding,dingline,instr1);
      }
      chkpref(nprf,2);
      if (thisline) {
        tag=1;
        strip(rest);
        thisindent=thisindent+strlen(instr1)-strlen(rest);
        lastindent=thisindent;
      }
      if (nextline) {
        strip(nrst);
        nextindent=strlen(instr2)-strlen(nrst);
      }
      if ((llc==0)&(thisline)) {
        llc++;
        if (thisnlist) {
          strcpy(outstr,":nl.\n");
          type[llc]='N';
        } else {
          strcpy(outstr,":ul.\n");
          type[llc]='U';
        }
        posllc[llc]=thisindent;
        fputs(outstr,outfile);
        thislen=0;
      }
      if (thisline) {
        if (thisnlist) {
          strcpy(outstr,":li.");
          strip(rest);
          strcat(outstr,rest);
        } else {
          strcpy(outstr,":li.");
          strip(dingline);
          strcat(outstr,dingline);
        }
        fputs(outstr,outfile);
        thislen=0;
      } else {
        /* Is it the beginning of a new paragraph?  */
        strcpy(outstr,instr1);
        strip(outstr);
        lastlen=thislen;
        thislen=strlen(outstr);
        /* if ((lastlen==0)||((thislen-lastlen>10))) { */
        if (blankbef>0) {
          strcpy(rest,outstr);
          strcpy(outstr,":p.");
          strcat(outstr,rest);
          fputs(outstr,outfile);
        } else {
          fputs(outstr,outfile);
        }
      }
      if (((thisline)&(!nextline)&(nextindent>=posllc[llc]))
        ||((contline==1)&(!nextline)&(nextindent>=posllc[llc]))
```

```
  ||((contline==1)&(nextline)&(nextindent==posllc[llc]))) {
  contline=1;
  continue;
}
if ((!nextline)&(nextindent>=posllc[llc])) continue;
if ((!thisline)&(nextline)) {
  llc++;
  if (nextnlist) {
     strcpy(outstr,":nl.\n");
     type[llc]='N';
  } else {
     strcpy(outstr,":ul.\n");
     type[llc]='U';
  }
  posllc[llc]=nextindent;
  fputs(outstr,outfile);
  thislen=0;
  continue;
}
if ((thisline)&(!nextline))
  while (nextindent<posllc[llc]) {
    contline=0;
    if (type[llc]=='N') {
      strcpy(outstr,":enl.\n");
    } else if (type[llc]=='U') {
      strcpy(outstr,":eul.\n");
    }
    llc--;
    lastindent=posllc[llc];
    fputs(outstr,outfile);
    thislen=0;
    continue;
}
if (((!thisline)&(!nextline)&(contline==1))) {
  contline=0;
  if (type[llc]=='N') {
    strcpy(outstr,":enl.\n");
  } else if (type[llc]=='U') {
    strcpy(outstr,":eul.\n");
  }
  fputs(outstr,outfile);
  thislen=0;
  llc--;
  lastindent=posllc[llc];
  continue;
}
if ((thisline)&(nextline)) {
  if (thisindent>nextindent) {
    while (nextindent<posllc[llc]) {
    contline=0;
    if (type[llc]=='N') {
      strcpy(outstr,":enl.\n");
    } else if (type[llc]=='U') {
      strcpy(outstr,":eul.\n");
    }
    llc--;
    lastindent=posllc[llc];
    fputs(outstr,outfile);
    thislen=0;
    continue;
    }
  }
  if (thisindent<nextindent) {
    llc++;
    if (nextnlist) {
      strcpy(outstr,":nl.\n");
      type[llc]='N';
    } else {
      strcpy(outstr,":ul.\n");
      type[llc]='U';
    }
    posllc[llc]=nextindent;
    fputs(outstr,outfile);
    thislen=0;
  }
    }

    strip(instr1);
    strcpy(outstr,instr1);

    /* If it's not been tagged and there's a blank
       line in front, then it's a paragraph. */
    if (tag==0) {
      if (blankbef>0) {
        strcpy(outstr,":p.");
        strcat(outstr,instr1);
      }
      /* and in any case, output the line */
      fputs(outstr,outfile);
    }

  }
  while (llc>0) {
    if (type[llc]=='N') {
      strcpy(outstr,"enl.\n");
    } else if (type[llc]=='U') {
      strcpy(outstr,"eul.\n");
    }
    fputs(outstr,outfile);
    llc--;
  }
  if (isexamp==2) fputs(":exmp.\n",outfile);

  /* Be tidy */
  fclose(infile);
  fclose(outfile);

}

/* Get next two non-blank lines.
   Return end=Y if eof on first line.
   Ignore "=-=-=-=-=" and following copyright lines
   Return count of blank lines before first line and
   between the two in blankbef and blankbet */

int gettwo(char *in1,char *in2)
{
  char *rc;
  int i,j,length;

  if (in1[0]=='\0') {
    /* Must be start of program */
    blankbet=-1;
    while ((in2[0]=='\n')||(in2[0]=='\0')) {
      blankbet++;
      rc=fgets(in2,1023,infile);
      if (rc==NULL) {
        return 1;
      }
    }
  }
  if (in2[0]=='\0') {
    /* Can only have been from an EOF last time. */
    return 1;
  }
  strcpy(in1,in2);
  blankbef=blankbet;
  blankbet=-1;
  in2[0]='\0';
  while (((in2[0]=='\n')||(in2[0]=='\0'))
           ||((in2[0]==' ')&(in2[1]=='\n'))) {
    blankbet++;
    if ((isexamp!=0)&(blankbef!=0)) fputs("\n",outfile);
    rc=fgets(in2,1023,infile);
    if (rc==NULL) break;
    if ((in2[1]=='=')&(in2[2]=='-')
         &(in2[3]=='=')&(in2[4]=='-')) {
      rc=fgets(in2,1023,infile);
      rc=fgets(in2,1023,infile);
    }
```

```
  }
  if (in2[0]=='\n') in2[0]='\0';
  /* Calculate indents */
  for (i=0;i<strlen(in1);i++) {
    thisindent=i;
    if (in1[i]!=' ') break;
  }
  for (i=0;i<strlen(in2);i++) {
    nextindent=i;
    if (in2[i]!=' ') break;
  }
  for (i=0;i<strlen(in1);i++) {
    if (in1[i]=='&')  {
      length=strlen(in1);
      for (j=length;j>i;j--) {
        in1[j+4]=in1[j];
      }
      in1[j++]='&';
      in1[j++]='a';
      in1[j++]='m';
      in1[j++]='p';
      in1[j]='.';
    }
  }
  for (i=0;i<strlen(in1);i++) {
    if ((in1[i]==':')
         &!((in1[i+1]==' ')
         ||(in1[i+1]=='\n')))  {
      length=strlen(in1);
      for (j=length;j>i;j--) {
        in1[j+4]=in1[j];
      }
      in1[j++]='&';
      in1[j++]='g';
      in1[j++]='m';
      in1[j++]='l';
      in1[j]='.';
    }
  }
  return 0;
}

/* Strip leading and trailing blanks from string */
int strip(char *string)
{
  int i,end;
  int start=-1;
  int indent=0;
  end=strlen(string);

  for (i=0;i<strlen(string);i++) {
    if (string[i]!=' ') break;
    indent++;
    start=i;
  }
  for (i=(strlen(string)-2);i>0;i--) {
    if (!((string[i]==' ')||(string[i]=='\n')))
break;
    end=i;
  }
  if ((start!=-1)||(end!=strlen(string))) {
    for (i=0;i<=(end-start-2);i++) {
      string[i]=string[i+start+1];
    }
    string[i]='\0';
  }
  return indent;
}

/* Check if the line is a header from numeric prefix.
   If so, write it out. */

char chkhd(char *line)
{
  char word[255];
  char rest[1024];
  char tstr[1024];
  char dot[2]=".";
  char head[3]=":h";
  int i,level,start;
  int test;

  getword(word,rest,line);

  /* Headers are only headers if the first word consists
     solely of numerics */
  test=strspn(word,"0123456789.");
  if (test!=strlen(word)) return 'N';

  /* If the first character of the first word is a ., then
     it isn't a header */
  if (word[0]=='.') return 'N';

  /* If the last character of the first word is a ., then
     it isn't a header */
  if (word[strlen(word)-1]=='.') return 'N';

  /* If two .s consecutive, then it's not a header */
  if (strstr(word,"..")!=NULL) return 'N';

  start=1;
  level=0;
  while (start>0) {
    level++;
    start=(int)(strchr(word,'.')-word);
    if ((start<0)||(start>strlen(word))) start=-1;
    if (start>0) word[start++]='x';
  }
  if (level==1) return 'N';
  if (level==2) {
    i=strlen(word)-1;
    if ((word[i]=='0')&(word[i-1]=='x')) level=1;
  }
  /* Before writing header, close off any existing lists */
  while (llc>0) {
    if (type[llc]=='N') {
      strcpy(outstr,"enl.\n");
    } else if (type[llc]=='U') {
      strcpy(outstr,"eul.\n");
    }
    fputs(outstr,outfile);
    llc--;
  }
  lastindent=baseindent;
  strip(rest);
  strcpy(tstr,dot);
  strcat(tstr,rest);
  itoa(level,word,10);
  strcpy(rest,word);
  strcat(rest,tstr);
  strcpy(tstr,head);
  strcat(tstr,rest);
  fputs(tstr,outfile);
  return 'Y';
}

/* Snarf first word from input line. */
void getword(char *word,char *rest,char *line)
{
  int i,j,flag;

  j=0;
  flag=0;
  word[0]='\0';
  rest[0]='\0';

  for (i=0;i<=strlen(line);i++) {
    if (!flag) {
      if (line[i]==' ') continue;
      else flag=1;
```

```
    }
    if ((line[i]==' ')&(flag==1)) {
      word[j]='\0';
      j=0;
      flag=2;
    }
    if (flag==1) word[j++]=line[i];
    else rest[j++]=line[i];
  }
  rest[j]='\0';
  strip(rest);
  return;
}

/* Initialize globals */
void init(void)
{
  deflist=0;
  termind=0;
  defind=0;
  revised=0;
  baseindent=4;
  lastindent=4;
  posllc[0]=baseindent;
  isexamp=0;

  return;
}

/* Check if line is part of definition list */
/* Return code 1 if line handled */
int chkdef(char *line1,char *line2)
{
  char stub1[1024];
  char stub2[1024];
  int pos1,pos2;
  int newlist;

  /* It's a definition list start if:
      there's a blank line before it,
      the line has at least three blanks separating
        two groups of text,
      the next line either starts lined with the second group,
        or lines up with the first, has at least two blanks
        before the second group, and lines up with that too. */

  /* If we're not already in a list,
                    check for start of new one */
  if (!deflist) {
    newlist=0;
    /* Not if no blank line */
    if (blankbef==0) return 0;
    strip(line1);
    strcpy(outstr,line1);
    /* Not if no sequence of at least 3 blanks */
    if (strstr(line1,"   ")==NULL) return 0;
    pos1=(int)(strstr(line1,"   ")-line1)+thisindent;
    /* Get the position of the definition term */
    substr(stub1,line1,pos1,strlen(line1)-pos1);
    strip(stub1);
    pos2=strlen(line1)-strlen(stub1)+thisindent-1;
    /* Where we look in line 1 is less because
                              we've stripped that */
    pos1=pos2-thisindent;

    /* Not if line 2 hasn't got blanks lining up
         with first line, and a nonblank after */
    if ((line2[pos1]==' ')&(line2[pos1+1]!=' ')
        &(line2[pos2+1]!=' ')) {
      deflist=1;
      termind=thisindent;
      defind=pos1+termind+1;
      newlist=1;
      lenterm=defind-termind;
      /* Definition term */
      substr(deft,line1,0,lenterm);
      strip(deft);
      /* Definition */
      substr(def,line1,defind-termind,
                  strlen(line1)-defind+termind);
      strip(def);
      fputs(":dl.\n",outfile);
      strcpy(outstr,":dt.");
      strcat(outstr,deft);
      strcat(outstr,"\n");
      fputs(outstr,outfile);
      strcpy(outstr,":dd.");
      strcat(outstr,def);
      fputs(outstr,outfile);
      return 1;
    } else {
      return 0;
    }
  }
  /* We are supposedly already in a definition list.
     Either the definition is continuing (if indent=defind),
     or there is a new item (indent=termind, and spaces before
     defind, but defind itself is non-blank,
     or there is a subsidiary element (indent> defind)
     or the list has ended. */
  if (thisindent==defind) {
    strip(line1);
    fputs(line1,outfile);
    return 1;
  } else if ((thisindent==termind)&(line1[defind]!=' ')
              &(line1[defind-1]==' ')) {
    /* Definition term */
    substr(deft,line1,termind,lenterm);
    strip(deft);
    /* Definition */
    substr(def,line1,defind,strlen(line1)-defind);
    strip(def);
    strcpy(outstr,":dt.");
    strcat(outstr,deft);
    strcat(outstr,"\n");
    fputs(outstr,outfile);
    strcpy(outstr,":dd.");
    strcat(outstr,def);
    fputs(outstr,outfile);
    return 1;
  } else if (thisindent>defind) {
    return 0;
  } else {
    defind=0;
    fputs(":edl.\n",outfile);
    deflist=0;
    return 0;
  }
  return 0;
}

/* Substring routine */
void substr(char *target,char *source,int start,int length)
{
  int i;

  for (i=0;i<length;i++) {
    target[i]=source[i+start];
  }
  target[length]='\0';
  return;
}

/* Return code 1 means line handled */
int revex(char *line1,char *line2)
{
  int i;
  char tch;

  /* Do we switch revisions on or off ? */
```

```
  if (line1[2]=='|') {
    if (!revised) {
      /* Set revision on */
      revised=1;
      fputs(":rev refid=revname.\n",outfile);
      line1[2]=' ';
    } else {
      line1[2]=' ';
    }
    for (i=0;i<strlen(line1);i++) {
       thisindent=i;
       if (line1[i]!=' ') break;
    }
  } else {
    if (revised) {
      /* Set revision off */
      revised=0;
      fputs(":erev refid=revname.\n",outfile);
    }
  }
  /* And anything in big braces is an example */
  if ((line1[4]=='├')&(line1[5]=='-')) {
    isexamp=1;
    fputs(":xmp.\n",outfile);
    return 1;
  } else if ((line1[4]=='Î')&(line1[5]=='-')) {
    isexamp=0;
    fputs(":exmp.\n",outfile);
    return 1;
  }
  if ((isexamp==1)||(isexamp==2)) {
    fputs(line1,outfile);
    return 1;
  } else {
    /* Finally, anything starting with character graphic box
       is an cgraphic */
    tch=line1[thisindent];
    if ((tch=='├')||(tch=='|')||(tch=='Î')||(tch=='Ø'))
{
      if (!isexamp) {
        isexamp=3;
        fputs(":cgraphic.\n",outfile);
      }
      fputs(line1,outfile);
      return 1;
    }
  }
  tch=line1[thisindent];
  if ((isexamp==3)&!((tch=='├')||(tch=='|')
                   ||(tch=='Î')||(tch=='Ø'))) {
    isexamp=0;
    fputs(":ecgraphic.\n",outfile);
    return 0;
  }
  return 0;
}

/* Check prefix to see if it's a list. */
void chkpref(char *string,int which)
{
  char ding[40];
  char dingline[1024];
  int length;

  if (strlen(string)==0) {
    if (which==1) thisline=0;
    else nextline=0;
    return;
  }
  if (which==1) thisnlist=0;
  else nextnlist=0;

  length=strlen(string);
  if ((length==1)&(verify(string,dinglist))) {
    if (which==1) thisline=1;
    else nextline=1;
    return;
  }
  if (which==1) thisnlist=1;
  else nextnlist=1;
  if (length>5) {
    if (which==1) thisline=0;
    else nextline=0;
    return;
  }
  if (verify(string,"ivx.")) {
    if (which==1) thisline=1;
    else nextline=1;
    return;
  }
  if (verify(string,"IVX.")) {
    if (which==1) thisline=1;
    else nextline=1;
    return;
  }
  if (verify(string,"0123456789.")) {
    if (which==1) thisline=1;
    else nextline=1;
    return;
  }
  if ((length==2)&(string[1]='.')) {
    if (which==1) thisline=1;
    else nextline=1;
    return;
  }
  if (which==1) thisline=0;
  else nextline=0;
  if (which==1) thisnlist=0;
  else nextnlist=0;
  return;
}


int verify(char *haystack,char *needle)
{
  int i,j,l1,l2,found,rc;
  rc=1;
  l1=strlen(needle);
  l2=strlen(haystack);
  for (i=0;i<l2;i++) {
    for (j=0;j<l1;j++) {
      found=0;
      if (haystack[i]==needle[j]) {
        found=1;
        break;
      }
    }
    if (found) continue;
    rc=0;
    break;
  }
  return rc;
}
```

A.4 Combine Image Cells in PSEG

With some versions of the IBMAFP driver to create AFPDS from workstation programs, PSEGs are created containing multiple image cells in the form of horizontal bands. The reason for this is that it is easier to deal with relatively small image blocks on the PC.

This is not a problem in many cases, but with IBM BookMaster and IBM BookManager, the PSEGs may not be processed appropriately. In IBM BookMaster, the whole of the PSEG is used, but the dimensions of the first cell are used for layout, so text may be overrun. In IBM BookManager, only the first cell is recognized.

The EXEC below will take such PSEGs, and recombine them into a single image cell.

It does this by throwing away all Begin and End Image structured fields except the first begin and the last end, and all Image Input Descriptors but the first (it is the Image Input Descriptor that defines the size of the image cell), and all Image Output Controls but the first (these define the origin from which all subsequent raster data is measured). Image Output Controls after the first have their origin added to the offset position from which the raster data is measured in each data record. Finally, in a second pass, the remaining Image Input Descriptor is updated with the total size of the PSEG.

A.4.1 Sample Code: Recombine PSEG Cells

```
/*  VM REXX EXEC.
    Reformat PSEG to single image cell.
    Note that this EXEC performs absolutely no error checking.
    This EXEC is designed for the situation where
    multiple image objects are in a single PSEG, and they
    are tiled vertically (i.e., they are horizontal bands
    across the total field.
    If you have other situations, this should at least be
    a starting point to build on. */

trace o
arg fn ft fm .
ofn=fn
oft=TEMP
ofm=fm

'STATE ' fn ft fm
if rc¬=0 then exit
'FINIS ' fn ft fm
'ERASE ' ofn oft ofm
nrecs=0
iocfirst=1
iidfirst=1
bimfirst=1
xdelta=-1
ydelta=-1

do forever
  'EXECIO 1 DISKR ' fn ft fm '  ( VAR INSTR'
   if rc¬=0 then leave
   nrecs=nrecs+1
   parse var instr 4 code +3 rest
   if code='d3a77b'x then do
      /* Structured Field is IOC - Image Output Control */
      /* Write first out unchanged, and set origins.
         Subsequent ones, increment deltas but don't write out */
      parse var rest 4 xorg +3 yorg +3 remains
      if iocfirst=1 then do
         xdelta=0
         ydelta=0
         xorigin=c2d(xorg)
         yorigin=c2d(yorg)
        'EXECIO 1 DISKW ' ofn oft ofm ' 0 ( VAR INSTR'
         iocfirst=0
         end
      else do
         xdelta=c2d(xorg)-xorigin
         ydelta=c2d(yorg)-yorigin
         end
    end
  else if code='d3ac7b'x then do
     /* Structured field is ICP - Image Cell Position */
     /* Add delta values to cell offsets, then write out */
     parse var instr start 10 xoff +2 yoff +2 remains
     xoffset=c2d(xoff)
     yoffset=c2d(yoff)
     xoffset=xoffset+xdelta
     yoffset=yoffset+ydelta
     xoff=d2c(xoffset,2)
     yoff=d2c(yoffset,2)
     rout=start||xoff||yoff||remains
    'EXECIO 1 DISKW ' ofn oft ofm ' 0 ( VAR ROUT'
    end
  else if code='d3a87b'x then do
     /* Structured field is BIM - Begin Image - IM */
     /* Write first of these out, ignore all others */
     if bimfirst=1 then do
       bimfirst=0
       'EXECIO 1 DISKW ' ofn oft ofm ' 0 ( VAR INSTR'
       end
     end
  else if code='d3a67b'x then do
     /* Structured field is IID - Image Input Descriptor */
     /* Write first of these out, use others to increment
        total y size of cell */
     parse var instr start 28 xsz +2 ysz +2 remains
     xsize=c2d(xsz)
     ysize=c2d(ysz)
     if iidfirst=1 then do
       iidfirst=0
```

```
        'EXECIO 1 DISKW ' ofn oft ofm ' 0 ( VAR INSTR'
         toty=ysize
         end
       else do
         toty=toty+ysize
         end
       end
    else if code='d3a97b'x then do
       /* Structured field is EIM - End Image - IM */
       /* All of these ignored - last one written before EPS */
       lastrec=instr
       end
    else if code='d3a95f'x then do
       /* Structured field is EPS - End Page Segment */
       /* Write unchanged preceded by last End Image */
      'EXECIO 1 DISKW ' ofn oft ofm ' 0 ( VAR LASTREC'
      'EXECIO 1 DISKW ' ofn oft ofm ' 0 ( VAR INSTR'
      end
    else do
      /* All other records passed unchanged */
      'EXECIO 1 DISKW ' ofn oft ofm ' 0 ( VAR INSTR'
      end
end
'FINIS ' ofn oft ofm
```

```
/* Pass two, to set total image size */

ft='TEMP'
oft='OUTPUT'
'ERASE ' ofn oft ofm

do forever
  'EXECIO 1 DISKR ' fn ft fm '  ( VAR INSTR'
   if rc¬=0 then leave
   parse var instr 4 code +3 rest
   if code='d3a67b'x then do
      /* Structured field is IID - Image Input Descriptor */
      parse var instr start 28 xsz +2 ysz +2 remains
      ysz=d2c(toty,2)
      rout=start||xsz||ysz||remains
     'EXECIO 1 DISKW ' ofn oft ofm ' 0 ( VAR ROUT'
      end
    else do
     'EXECIO 1 DISKW ' ofn oft ofm ' 0 ( VAR INSTR'
      end
end
'ERASE ' fn ft fm
'FINIS ' ofn oft ofm
```

A.5 Reblock Uploaded PSEG

Some versions of the IBMAFP driver to create AFPDS from workstation programs are provided with an EXEC to reblock files uploaded from the workstation. (This is necessary because the variable record file structure of PSEG files is not readily handled by most file transfer programs.)

Where an EXEC or CLIST is provided, this should be used. Where it is not, the EXEC below can be used.

The file should have been uploaded as a variable length record format file (RECFM V) with a maximum record length of 32767.

We also found that early versions of the AFP Workbench for Windows program didn't provide the leading '!' in all cases, so we have commented three places in the EXEC where it can be modified to handle that situation if you come across it.

A.5.1 Sample Code: Reblock Uploaded PSEG

```
/*  VM REXX EXEC.
   Splits uploaded file into records
*/
trace o
arg fn1 ft1 fm1 fn2 ft2 fm2

if fn1 = '' | fn1 = '?' then
do
 say 'AFPSPLIT takes an unformatted AFPDS file and   '
 say 'formats it into records, each record consisting'
 say 'of a structured field.                         '
 say '                                               '
 say 'Format:                                        '
 say '                                               '
 say ' AFPSPLIT  fn1 ft1 fm1 fn2 ft2 fm2             '
 say '                                               '
 say ' fn1 ft1 fm1                                   '
 say '     The name of the file to be formatted.     '
 say '               fm1 = A                         '
 say ' fn2 ft2 fm2                                   '
 say '     The name of the output formatted file.    '
 say '     Defaults: ft2 = LIST3820                  '
 say '               fm2 = A                         '
 say '                                               '
 exit
end

if fn1='' | ft1='' then do
   say 'Error - No input file given'
   exit
end
if fm1='' then fm1='A'
if fn2='' then fn2=fn1
if ft2='' then ft2='LIST3820'
if fm2='' then fm2='A'

          /* CHECK WHETHER FILE EXISTS */
ADDRESS COMMAND 'ESTATE' fn1 ft1 fm1
if rc ¬= 0 then
          /* woops -- file gone? */
  do
    SAY 'AFPSPLIT: File' fn1 ft1 fm1 'not found; RC='RC
    exit
  end

'listfile ' fn1 ft1 fm1 ' (stack alloc '
```

```
pull p1 p2 p3 recfm lrecl norecs blocks .
code=0
if recfm¬='V'  then do
  say 'Record format is not V for Variable.'
  code=1
end
if lrecl¬=32767 then do
  say 'Record length is not 32767.'
  code=1
  if norecs=1 then do
    say "- but only one record, so that's cool."
    code=0;
  end
end
if code=1 then do
  say 'Upload again with correct parameters.'
  exit
end

infile = fn1 ft1 fm1
outfile = fn2 ft2 fm2
'erase 'outfile

line = ''
bytes = 0
do forever
   'execio 1 diskr 'infile' (VAR INLINE'
   if rc¬=0 then leave
   line = line||inline

   do forever
      /*  If the file doesn't contain leading ! characters
          in every record, then the following line should be
          replaced by
          linelen = c2d(substr(line,1,2))
      */
          linelen = c2d(substr(line,2,2))

          if linelen>length(line) then leave

          bytes = bytes + linelen

          /* If the file doesn't have leading ! characters,
             this time comment out the following "if" block */
          if substr(line,1,1)¬='!' then do
             say 'Error at 'bytes' bytes - attempting recovery'
             line = substr(line,2)
             iterate
          end

          /* If the file doesn't have leading ! characters,
             replace the following line by
          outline = substr(line,1,linelen)
          */
          outline = substr(line,1,linelen+1)
          'execio 1 diskw 'outfile' (VAR OUTLINE'

          /* If the file doesn't have leading ! characters,
             replace the following line by
          line = substr(line,linelen+1)
          */
          line = substr(line,linelen+2)
     end
end
'finis * * *'
say 'Split completed'
exit
```

Appendix B. Images, Graphics, and Data Streams

B.1 Major Image Formats and Where They Are Used

Table 5. Major Image Formats and Where They Are Used. This table indicates, for the major image formats, where they are used and what tools either generate them or accept them as input.

Major Image Formats	Used by	Tools that create	Tools that accept
IOCA..	*ImagePlus* *GDDM* *OV/2 Release2 Editor* *OV/400 Editor* *DW 5/2* *DW /370*	*PC/IDU* *OIS* *DW 5/2* *DW/370* *GDQF*	*PSF* *PSF/2* *PC/IDU* *OIS* *DW/370* *GDQF*
PSEG	*IBM BookMaster* *IBM BookManager* *SGML Translator*	*PC/IDU* *GDDM-IVU* *ProcessMaster* *GDQF* *ImagePlus*	*GDDM-IVU* *ProcessMaster* *GDQF* *ImagePlus*
EPS	*IBM BookMaster* *SGML Translator*	*ProcessMaster* *DW 5/2* *PostScript*	*DrawMaster** *ProcessMaster*
ADMIMG	*DW/370* *IBM BookManager*	*GDDM-IVU* *ProcessMaster* *DW/370* *GDQF*	*GDDM-IVU* *ProcessMaster* *DW/370* *GDQF*
TIFF	*GDDM* *Workstation* *Applications*	*PC/IDU* *OIS* *ProcessMaster* *DW 5/2* *GDQF*	*PC/IDU* *OIS* *ProcessMaster* *DW 5/2* *GDQF*

Note:
PSEG/IM1 = normal bitmap
IOCA = Compressed image

Note:
PostScript interpreter accepts normal PostScript as well as EPS.
An EPS file actually can contain image and/or graphic and/or text.

Note:
PMWE supports ADMIMG, Host ProcessMaster does not support ADMIMG.

B.2 Major Graphics Formats and Where They Are Used

Table 6. Major Graphics Formats and Where They Are Used. This table indicates, for the major graphics formats, where they are used and what tools either generate them or accept them as input.

Major Graphics Formats	Used by	Tools that create	Tools that accept
RFT/GOCA	*OV/2 Release2 Editor* *OV/400 Editor* *DW 5/2* *DW /370*	*DW/370* *Display Graphics* *DW 5/2* *OV/400 Editor*	*DW/370* *Display Graphics* *DW 5/2* *OV/400 Editor*
EPS	*IBM BookMaster* *SGML Translator*	*ProcessMaster* *DW 5/2* *PostScript Interpreter*	*DrawMaster* *ProcessMaster*
ADMGDF	*DW /370* *IBM BookManager*	*GDDM* *GDQF* *DW/370* *DrawMaster* *ProcessMaster*	*GDDM* *GDDM-ICU* *GDQF* *AS* *DW/370* *DrawMaster* *ProcessMaster* *Workstation Applications*

Note:
PostScript interpreter accepts normal PostScript as well as EPS
An EPS file actually can contain image and/or graphic and/or text

B.3 Major Data Streams and Where They Are Used

Table 7 (Page 1 of 3). Major Data Streams and Where They Are Used. This table indicates, for the major data streams where they are used and what tools either generate them or accept them as input. A more detailed overview is contained inD.1, "Product and Operating System Tables" on page 225 .

Major Data Streams	Tools that create	Tools that accept	Hardware working with it
MO:DCA **FOCA** **IOCA** **PTOCA** **GOCA** **BCOCA**	*ImagePlus*	*ImagePlus* *PSF* *PSF/2* *Print Services/400*	

Table 7 (Page 2 of 3). Major Data Streams and Where They Are Used. This table indicates, for the major data streams where they are used and what tools either generate them or accept them as input. A more detailed overview is contained in D.1, "Product and Operating System Tables" on page 225.

AFPDS **FOCA** **IOCA** **PTOCA** **GOCA** **BCOCA**	*DCF* *BookManager* *GDQF* *GDDM* *OGL* *AS/400 AFP Utilities* *ImagePlus* *AFPDS-driver installed under Windows* *Line to AFP convert program* *Complementary products*	*PSF* *PSF/2* *ImagePlus* *BrowseMaster* *GDQF* *GDDM*	
IPDS	*PSF* *Print Services/400* *PSF/2*		*all AFP printers*
RFT:DCA	*DW/370* *Office Vision/MVS* *Office Vision/400* *Office Vision/2*	*GDDM*	*all printers that have GDDM as device driver available*
CDRA	*DW/370* *Office Vision/MVS* *Office Vision/400* *Office Vision/2*	*GDDM*	*all printers that have GDDM as device driver available*
DIA	*DW/370* *Office Vision/MVS* *Office Vision/400* *Office Vision/2*	*GDDM*	*all printers that have GDDM as device driver available*
FD:OCA	*DW/370* *Office Vision/400*	*GDDM*	*all printers that have GDDM as device driver available*
3270DS	*traditional application programs*	*traditional application programs*	*3270 Display Stations*
1403DS	*traditional application programs*	*traditional application programs* *PSF* *PSF/2* *Print Services/400* *traditional line printers*	*Line printers* *AFP printers*
PostScript	*DCF* *Workstation applications running under Windows*	*DCF* *PostScript adapter program* *PSF/2 V1.1*	*PostScript printers*

Table 7 (Page 3 of 3). Major Data Streams and Where They Are Used. This table indicates, for the major data streams where they are used and what tools either generate them or accept them as input. A more detailed overview is contained in D.1, "Product and Operating System Tables" on page 225.			
PPDS	*Workstation applications*		*Workstation printers on LPT1 supported by PSF/2*
PCL	*Workstation applications*		*Workstation printers on LPT1 supported by PSF/2*
Note: Object Content Architectures are part of AFPDS and as such they are generated by the programs generating AFPDS. For more information please refer to *Information Interchange Architecture: Concepts, GG24-3503.*			
Note: Input to PSF is 1403DS or AFPDS. Input to PSF/2 is ASCII, Meta File, AFPDS, and PostScript.			
Note: 1403DS must be converted to ASCII, PostScript, or AFPDS prior to input to PSF/2.			
Note: In order to generate PPDS or PCL data streams the corresponding printer driver must be available on the workstation.			

B.4 Types of Bitmap Images

B.4.1 Bilevel Images

Bilevel images are images that contain only two values, black and white. Line drawings are typical bilevel images.

Bilevel images are the most economical image types for storage, as each pixel can be represented by a single bit.

Most computer printers are only capable of printing bilevel images. This means that other images have to be simulated by bilevel. A typical method is to use halftoning to represent a grayscale image. Color images are rendered by assigning grayscales to colors before halftoning.

B.4.2 Grayscale Images

Grayscale images use more than a single bit per pixel to store information about the relative brightness of the picture element.

Typically, 4 or 8 bits are used, allowing the representation of either 16 or 256 gray levels.

What each level represents is application dependent. Some images represent black at one end of the scale and white at the other, with roughly evenly split light levels between, while more sophisticated ones attempt to give more information about the image by making the number of pixels in the image assigned to each gray level approximately equal in number. In the latter case, a "gray response curve" is then necessary to interpret the image correctly.

A common technique to do this assignment is called "histogram equalization."

B.4.3 Palette-color Images

In this type of image, the information stored for each pixel is an index into a "palette," or color look-up table.

Images with limited color information are often stored this way.

B.4.4 RGB Images

An RGB image is capable of storing more or less continuous color information. Here, each pixel is represented by three numbers, representing the relative strengths of the red, green, and blue components of the light value for the pixel.

Occasionally other values per pixel are stored, to represent such values as opacity or transparency.

RGB images are device dependent; that is, the precise color rendered by the display, printer, or other output device is not uniquely defined in the absence of information about the device. The same triplet value may render different results on different devices.

In prepress and other high-quality work, it is necessary to define the color precisely. In an RGB system, this can only be done by specifying other information besides the pixel values, such as the chromaticities of the reference white value and each individual primary color, top and bottom range limits for each primary (headrooms and footrooms), and transfer functions for each primary (to map response curves to a linear representation). This is called *colorimetry* information.

B.4.5 CMYK Images

Color image encoding using the CYMK model is essentially designed for color printing. CYMK images are device-specific, and for each pixel, they define the amount of cyan, magenta, yellow, and black process inks that are to be used for that pixel.

This model is not suited for interchange.

B.4.6 YCbCr Images

The television industry doesn't use RGB; for digital video, they define *luminescence* and *chrominance* values of the image.

For this kind of work, the YCbCr standard, based on CCIR Recommendation 601-1, Encoding Parameters of Digital Television for Studios, is used, for example as part of the TIF standard.

Each triplet is defined as a Y luminescence value, and two color difference chrominance components, Cb and Cr. The image must also have defined for it coefficients for transfer functions, and reference black and white values, to enable transform to an RGB color space.

B.4.7 CIE L*a*b* Images

CIE L*a*b* is a standard for the definition of high-quality color images. Unlike RGB, **CIE L*a*b*** unambiguously defines a color space which is inherently colorimetric. It is thus completely device independent, and is based on the CIE 1931 Standard Observer, which means that it is designed to match human color response.

The color space of the **CIE L*a*b*** model is three dimensional, one axis being L* representing lightness, a* representing the red/green axis, and b* the yellow/blue axis.

Besides being colorimetric, separating the color space in this way allows simple conversion to monochrome, and compressibility is better than RGB.

Appendix C. Products

The following pages contain overview information in tabular form. They are intended to help a reader to find quick answers to questions such as:

- What data formats should I use in my environment?
- What programs should I use to get the data formats I need?
- Do I have all required programs available?
- What data stream is required to proceed?
- What programs accept the data I have?
- What program generates the output format needed?
- If I need to convert, what programs do I need?

C.1 Software for Print and View Data Streams under MVS and VM

Table 8. Software for Print and View Data Streams. This table indicates which products work with print and view data streams in SAA environments.

Operating System	Program product	Remarks
MVS	*OfficeVision/MVS*	
	DW/370	
	ImagePlus Folder Application Facility	
	ImagePlus Object Distribution Manager	
	IBM BookMaster	
	BrowseMaster	
	DCF	
	GDDM	
	ProcessMaster	
	SGML Translator	*SGML is also processed under DEC** and UNIX*
	PostScript to AFPDS	*works under ProcessMaster*
	AFP Software	*PSF, PMF, PPFA, OGL, Complementary solutions*
	TextTagger/ESA	*not announced in all countries*
	IBM BookManager READ	
	IBM BookManager Build	
VM	*OfficeVision/VM*	*(PROFS)*
	IBM BookMaster	
	DW/370	
	BrowseMaster	
	DCF	
	GDDM	
	ProcessMaster	
	SGML Translator	*SGML is also processed under DEC and UNIX*
	TextTagger	*not announced in all countries*
	PostScript to AFPDS	*works under ProcessMaster*
	AFP Software	*PSF, PMF, PPFA, OGL, Complementary solutions*
	IBM BookManager READ	
	IBM BookManager Build	

C.2 Software for Print and View Data Streams under OS/400 and OS/2

Table 9. Software for Print and View Data Streams. This table indicates which products work with print and view data streams in SAA environments.

Operating System	Program product	Remarks
AS/400	*OfficeVision/400*	
	ImagePlus Workfolder Application Facility/400	
	AFP Utilities/400	*PFU, RSU, RMU*
	BGU	
Workstation	*OfficeVision/2 LAN*	
	ImagePlus Workstation Program/DOS	
	ImagePlus Workstation Program/2	
	AFP Workbench for Windows	*Reads AFP files, prints and files using workstation printer drivers*
	ProcessMaster Workstation Edition	
	TextTagger OS/2	*creates SGML files as well as BookMaster*
	IBM BookManager READ /DOS	
	IBM BookManager READ OS/2	
	DW 5/2	
	PSF/2	
	AFP Driver	*Part of PSF/2 and AFP Workbench for Windows, drivers for OS/2 and DOS/Windows*
	DTP programs	*AFPDS driver installed under OS/2.* *AFPDS driver installed under DOS/Windows* *PostScript as input to PSF/2*

C.3 Input and Output of Major Programs

Table 10 (Page 1 of 2). Inputs and Outputs of Major Programs. This table shows the main input and output formats available with several of important products.

Program	Input	Output
GDDM	*AFPDS* *CCITT G.3* *CCITT G.4* *IOCA* *IMAGE* *MO:DCA* *IMDS* *ADMIMG* *RFT* *BITMAP* *GDF* *CGM* *GL*	*AFPDS* *MO:DCA-P* *IOCA* *OS/2 Bitmap* *(Metafile)* *RFTDCA* *ADMIMG* *ADMGDF* *TIFF* *CGM*
GDQF	*AFPDS* *CCITT G.3* *CCITT G.4* *TIFF* *MO:DCA* *IMDS* *ADMIMG* *RFT* *BITMAP* *GDF* *GL*	*AFPDS* *MO:DCA-P* *IOCA* *OS/2 Bitmap* *(Metafile)* *RFTDCA* *ADMIMG* *TIFF* *CGM* *Gamma Fax*
PSF	*1403DS* *AFPDS*	*IPDS*
PSF/400	*1403DS* *SCS* *AFPDS*	*IPDS*
PSF/2	*ASCII* *Metafile* *AFPDS* *PostScript*	*IPDS* *PPDS* *HP-PCL4* *HP-PCL5*
RPM V.2.	*IPDS*	*IPDS*
RPMV.3	*IPDS* *ASCII*	*IPDS*
OGL	*3270DS* *as source code*	*AFPDS*
DCF	*3270DS* *as source code*	*AFPDS* *PostScript* *and many others*
ImagePlus	*IOCA*	*MO:DCA-P*

Table 10 (Page 2 of 2). Inputs and Outputs of Major Programs. This table shows the main input and output formats available with several of important products.

DW/370	*3270DS* *ADMGDF*	*RFT-DCA* *FFT-DCA* *SCS* *3270DS*
Office System Products	*3270DS* *ADMGDF*	*RFT-DCA* *FFT-DCA* *SCS* *3270DS*
Corel Draw	*TIFF* *EPS* *PIC* *PLT* *PIF* *and many others*	*TIFF* *EPS* *PIC* *PLT* *PIF* *PostScript* *AFPDS using IBMAFP driver* *and others*
Ami Pro	*ASCII* *DCA/FFT* *DCA/RFT* *DIF* *DOC* *RTF* *and many others*	*ASCII* *DCA/FFT* *DCA/RFT* *DIF* *DOC* *RTF* *PostScript* *AFPDS using IBMAFP driver* *and others*
OIS	*TIFF* *IOCA* *MOD:DCA-P* *Bitmap* *RFT:DCA* *PCX* *CALS* *PIC*	*IOCA* *MOD:DCA-P* *PCX* *PIC* *RFT:DCA* *TIFF* *Bitmap* *AFPDS using IBMAFP driver*

Note: PSF/2 and AFP Workbench for Windows contain **AFPDS Drivers**, which can be installed as a printer drivers under OS/2 and DOS/Windows. These can produce AFPDS output with most programs which run under OS/2 or DOS/Windows.
This AFPDS data stream can be printed on any AFP printer in any environment.

C.4 Access to Data Based upon Office

Table 11. Access to Data Based upon Office. This table shows how data may be moved in and out of various environments. Environments covered are Office, Publishing and ImagePlus.

Office Products	Output formats	Input formats	Transform tools
OfficeVision/2 LAN* *OfficeVision/MVS** *OfficeVision/VM** *OfficeVision/400**	*RFTDCA* *ASCII* *EBDCIC* *PRINT* *PDA*	*ASCII* *EBDCIC* *TIFF* *ADMIMG*	*ODF* *TextTagger* *OCR*

Note: Tools are required only for moving Office Text into Publishing

Note:
ODF = Office Document Feature
Accessed under ProcessMaster. Converts RFT into GML, SGML. Manual editing is required to achieve full conversion.
ODF accepts RFT:DCA input and generates

GML
    Input for IBM BookMaster
CALS SGML
    Input for IBM BookMaster

Note:
TextTagger analyzes a document and creates markup in the text. Manual editing is suggested to check for proper conversion.
Input formats accepted by TextTagger:

ASCII
EBCDIC
RFT:DCA
PRINT
PDA

Output formats generated by TextTagger:

GML
    Input for IBM BookMaster
    Input for IBM BookManager Build
SGML
    Input for SGML Translator
    Output of SGML Translator is input to IBM BookManager Build

Note:
OCR = Optical Character Recognition
PDA = Processed Document Architecture
There are several solutions available through IBM Business Partners which generate either ASCII or PDA, which in turn can be input to TextTagger.

C.5 Access to Data Based upon Publishing

Table 12. Access to Data Based upon Publishing. This table shows how data may be moved in and out of various environments. Environments covered are Office, Publishing and ImagePlus.

Publishing Products	Output formats	Input formats	Transform tools
Production Publishing *BookMaster* *BrowseMaster* *ProcessMaster* *ProcessMaster CALS* *Feature SGML* *Translator TextTagger* *BookManager* **Professional Publishing** *IBM Interleaf Publisher PS/2* *Interleaf PS/6000* **Personal Publishing** *PostScript Complementary Software*	*AFPDS* *CDPDS* *EBCDIC* *Print Spool File*	*GML* *SGML* *CALS SGML* *EBCDIC* *IOCA* *PSEG*	*ODF* *TextTagger* *OCR* *Transform programs of ImagePlus*
Note: Transform programs of ImagePlus are used to transform MO:DCA-P to either IOCA, or PSEG.			
Note: Other Tools for transforming to the appropriate data stream are: TextTagger PC/IDU OIS			

C.6 Access to Data Based upon ImagePlus

Table 13. Access to Data Based upon ImagePlus. This table shows how data may be moved in and out of various environments. Environments covered are: Office, Publishing and ImagePlus.

ImagePlus Products	Output formats	Input formats	Transform tools
MVS/ESA*			
ImagePlus Folder Application Facility	*MODCA* *IOCA* *ASCII*	*MODCA* *IOCA* *ASCII*	*ODF*
ImagePlus Object Distribution Manager			
AS/400			
ImagePlus Workfolder Application Facility/400			
Workstation			
ImagePlus Workstation Program/DOS			
ImagePlus Workstation Program/2			

Note: Transform programs are used to provide an appropriate input data stream to ImagePlus.

ASCII(Workstation)
IOCA (Workstation)
MO:DCA-P (MVS, AS/400)
EBCDIC (AS/400)

Note: Transform programs are used to provide an appropriate output data stream from ImagePlus for office or publishing.

IOCA (Publishing)
ASCII (Workstation)

C.7 Page Printer Formatting Aid/370

IBM Page Printer Formatting Aid/370 Release creates Advanced Function Printing (AFP) resources that control how information is printed on a page.

These resources are called form definitions (FORMDEFs) and page definitions (PAGEDEFs). FORMDEFs specify how the printer should handle the physical sheets of paper. PAGEDEFs specify how data should be arranged on the logical page.

These printing resources provide the capability to print in simplex or duplex mode, to print one or more application pages on the same side of a physical page, to print in portrait or landscape mode and to include or exclude application data and electronic overlays on different pages of the same print job. Several additional parameters can be specified to choose differing colors, print density, or paper bins on printers that support those capabilities.

PAGEDEFs and FORMDEFs defined with predecessor IBM PPFA programs are upwardly compatible with IBM PPFA/370.

PAGEDEFs and FORMDEFs created for use in the VM environment are functionally equivalent to PAGEDEFs and FORMDEFs created for use in the MVS and VSE environments.

Developing FORMDEFs and PAGEDEFs with PPFA on the main frame is to use a normal text editor and after entering the source code it is submitted to the compiler PPFA which generates a machine readable code for the printer driver. that can also be exchanged with other systems and other SAA platforms. Meanwhile some complementary software on work stations is available allowing a user to develop PAGEDEFs and FORMDEFs and overlays in a WYSIWYG mode.

PPFA/370 is a follow on product of preceding versions which were designed for the environments MVS, VM, or VSE. IBM Page Printer Formatting Aid/370 is a program that replaces the following licensed programs with a single, new, cross system product:

C.8 Overlay Generation Language/370

IBM Overlay Generation Language/370 Release 1.0 provides the ability to create electronic forms overlays for use by Print Services Facility in formatting print jobs for Advanced Function Printing (AFP) page printers. The program has been enhanced with improved algorithms for the formation of rounded corners when using dotted or dashed lines with OGL graphics commands.

OGL/370 has the ability to identify which IBM System/370 SCP environment it is operating in and conditionally process system dependent code without user intervention.

Generating overlays allows placement of all necessary information on the page that is not part of the output of an application program:

- Images
- Graphics
- Text

  - In selected fonts
  - Rotated as required
- Shaded areas
- Boxes lines
- Circles

Overlays are developed on the mainframe using a normal text editor. After the source code is entered, it is submitted to the OGL compiler, which generates a machine readable overlay (AFPDS, MO:DCA-R).

Complementary workstation software is available which allows the user to develop overlays in a WYSIWYG mode. PAGEDEFs and FORMDEFs can also be created by this software.

Users should also be aware that overlays can now be generated by virtually any graphics program on the workstation; the IBMAFP printer driver allows them to create overlays (see 13.5, "Scan a Logo, Improve It, and Integrate It in an Overlay" on page 111).

C.9 Line Data to AFPDS Converter

This is not a formal IBM product, but is generally available.

This program will convert System 370 line printer data streams to Advanced Function Print Data Streams (AFPDS) in a manner analogous to PSF/MVS. The output format is controlled by a Page Definition. The resulting AFPDS may be printed by any AFP-compatible printer.

C.9.1 Converting Line Data to AFPDS

Line data consists of individual records containing one line of text to be printed and optional *carriage control* and *table reference* characters (for identifying the font to print a line on a 3800 line printer).

When line data is to be printed on an AFP page printer, it is converted into printable pages by Print Services Facility (PSF). PSF uses a Page Definition (PAGEDEF) to construct each printed page. The Page Definition contains instructions on how a page is to be formatted, including the number of lines per page, the font(s) to use in printing each line, and the placement of lines on the page.

In addition to a Page Definition and line data, PSF can print other text and graphics on a page. By including *structured fields* within line data, PSF may be instructed to place a page segment on the same page with line data. The Include Page Segment (IPS) structured field names a segment contained in a segment library to be included on a page. For more information on using structured fields within line data, see *PSF/MVS User's Programming Guide, S544-3084*.

LINEAFP takes as input a PAGEDEF (and, optionally, a FORMDEF) and line data, and creates as output an AFP data stream that can print on any AFP-supported printer. In environments which do not process line data for AFP page printers, such as AS/400, the program may be used to convert MVS-created line data sets to be printed on AS/400 attached printers. LINEAFP currently runs in MVS/370,

MVS/XA*, or MVS/ESA. The output dataset is suitable for sending to any system that supports AFP data streams that contain no embedded line data.

C.10 Document Composition Facility

DCF is a host-based text formatting program designed for high-volume, in-house industrial, technical, and educational publishing and text processing applications.

DCF includes the SCRIPT/VS text formatter, which provides formatting support for the BookMaster, SGML Translator, and BookManager BUILD products.

A Foreground Environment Feature enables DCF to be used within TSO under MVS, and CICS* and CMS under VM.

With the Document Library Facility (DLF), the SCRIPT/VS formatter can operate in a batch environment under MVS and VSE. DLF provides an interface to the formatter.

DCF has an Office Document Feature (ODF) which allows DCF to convert a RFT:DCA file to a file suitable for formatting by SCRIPT/VS, and ODF can also convert this file back to an RFT:DCA file. This conversion is at a pure format level and should not be considered as a substitute for conversion to GML.

DCF has an optional Mathematical Formula Formatter (SMFF). This feature allows the entry of tags to define mathematical formulas, allowing them to be formatted by DCF for later printing or display.

DCF can now handle formatting of DBCS text in documents.

Highlights:

- Output for:
  - Line printers
  - AFPDS
  - PostScript
- Supports color
- Separates masters by color
- Includes many kinds of image:
  - Page Segments
  - IOCA uncompressed
  - IOCA compressed
  - EPS if output device is a PostScript device
- Interprets:
  - The IBM Script formatting language
  - GML tags
  - BookManager tags
- The output of DCF is the base for BookManger Build

C.11 SCRIPT Mathematical Formula Formatter Feature

The SCRIPT Mathematical Formula Formatter is a separately charged feature of DCF. It allows equations to be defined to DCF by using a descriptive formula language designed to be easy to learn and use. The mathematical expression can then be described in words, roughly equivalent to the way one would read it over the telephone.

Input can be made from any workstation or terminal used for DCF text. Since special symbols that appear in equations are entered as symbolic names, any alphanumeric keyboard can be used to create these equations. Users also have the ability to enter abbreviated symbolic names for both the special symbols used in the equations and the formatting keywords used to describe them, or to define their own symbols for complete formulas or parts of equations.

A Formula Formatter Starter Set of Generalized Markup Language (GML) tags is provided with SMFF that can be installed as an extension to the DCF starter set, or it can be used by other DCF applications or source documents. The IBM Publishing Systems BookMaster provides a tag and macro interface to use SMFF within BookMaster documents.

SMFF extends the function set of Document Composition Facility. It provides for typesetting of mathematical equations and scientific formulas. It is designed to be easy to use for people who know neither mathematics nor typography. SMFF does not check the mathematical validity of an expression. Equations can be interspersed within DCF text files, and can be merged with images such as signatures, logos, halftone photographs, vector graphics converted to images, or graphics scanned on a host attached scanner, or created by IBM programs designed for that purpose.

SMFF can format piles of characters, Greek letters, fractions, subscripts and superscripts, roots, big parentheses and brackets, embellished characters, and more. The printing of the formatted equations must be done on an all-points addressable output device, or page printer.

PostScript output is also supported by DCF and SMFF.

Note: Within VSE SMFF is not available. It is presently also not possible to generate PostScript output with DCF under VSE.

C.12 ProcessMaster CALS Application Feature

The CALS Application Feature is part of a set of products that address the requirements of the CALS initiative. It is composed of five components:

- CALS MIL-M-38784B Formatting Application
- CALS 1840A Tape Create/Read Utility
- CALS MIL-D-28003 CGM/ADMGDF Transform
- CALS MIL-R-28002 CCITT G4 Raster Image/PSEG Transform
- CALS OS/2 Workstation Driver

CALS formatting application provides the ability to format documents that comply with the Department of Defense using DCF. This will allow DCF to format compliant documentation for all of the supported APA-printers. In addition, the same documentation will be able to be printed on any DCF-supported printer for draft copies.

The SGML Translator DCF Edition will validate the source code and translate it into CALS DCF. Using the output of the translator and the CALS DCF macro library, DCF then creates MIL-M-38784B conforming documents. Graphic elements referenced by the SGML source must be in PSEG or PostScript format for the destination printer. Document objects in standard CALS graphic format must be transformed to PSEG or PostScript format before formatting.

Tape Generation Utility The Tape Generation Utility provides a means of writing a set of files or data sets comprising one or more MIL-STD-1840A documents onto tape in ANSI X3.27 format for delivery to the Department of Defense or other companies. It also provides a means of reading tapes written in that format.

Graphic Transforms The Graphic Transform converts vector graphic objects in MIL-STD-28003 CGM (Computer Graphics Metafile) format to and from ADMGDF format.

Image Transforms The Image Transform converts the MIL-R-28002 RASTER format to and from the PSEG format.

CALS OS/2 workstation driver The Workstation Driver function provides a set of menus and procedures that allow the user to access host and workstation document development tools in a CUA* Presentation Manager interface. Some of the functions accessible are:

- Host or workstation print
- Library functions
- Editors (text, graphic, and image)
- SGML validator and translator
- Graphic and image conversions
- TextTagger (host or workstation)
- Tape creation
  - Document Declarations
  - Build headers
  - Tape create
- Tape read

The CALS OS/2 Workstation Driver Feature is distributed with the CALS Application feature.

C.13 SGML Translator DCF Edition

The SGML Translator DCF Edition is a program that processes SGML documents prior to their formatting by the Document Composition Facility. It translates the SGML data into DCF compatible source data. Once created, this data can be processed by the Document Composition Facility to create printable information. The SGML Translator also validates that the data being processed conforms to the SGML standard (ISO International Standard 8879-1986).

The SGML Translator DCF Edition is made up of four components:

- SGML Validator
- DCF Translator
- Sample SGML Starter Application
- Mathematical Formula Application

SGML Validator The SGML Validator is a program that is used to validate SGML documents after they are created, edited, or received from another source. It parses the SGML source input according to the ISO 8879/1986 standard and insures the data complies with the standard. This SGML Validator may be used to process any SGML Document Type Definition and document developed in accordance with ISO International Standard 8879 and, as such, is a valuable tool to be used in developing Document Type Definitions and SGML documents. There is no output from this component other than messages indicating the results of the validation.

DCF Translator The DCF Translator converts SGML documents marked up according to the SGML Reference Concrete Syntax into DCF/GML syntax and provides for other preprocessing needed to use DCF as the formatter for SGML documents. It is used to support new SGML applications and applications migrated to SGML from existing DCF applications.

Sample SGML Starter Application This component consists of a Document Type Definition (DTD) and a translation table and a set of processing routines. These files provide a basic level of functions for simple applications and provide an example of an SGML application based on DCF. The DCF starter set maclib is used to process the data translated by this component.

Mathematical Formula Application A mathematical formula application is provided with the SGML Translator. This application is not intended to be used by itself, but can be incorporated into the SGML Starter Application or a customized application to allow users to mark-up mathematical formulas according to the standards defined in ISO Technical Report 9573.

C.14 SGML Text Write OS/2 Edition and SGML Text Write Tools OS/2 Edition

SGML Text Write is two major products, with the first a prerequisite for the second:

1. Text Write OS/2 Edition
2. SGML Text Write Tools OS/2 Edition

C.14.1 SGML Text Write OS/2 Edition Product Overview

The SGML Text Write OS/2 Edition is a software program that helps writers and editors create and modify SGML-compliant documents. The SGML Text Write OS/2 Edition also provides standard word processing functions. SGML Text Write OS/2 Edition runs in an OS/2 environment under Presentation Manager and takes advantage of the inherent features in this windowing environment.

SGML Text Write OS/2 Edition:

- Creates CALS-compliant and other SGML Documents that can be formatted on an IBM host using:
  Document Composition Facility IBM SGML Translator DCF Edition, Publishing Systems ProcessMaster CALS Application feature.
- Provides a WYSIWYG editing interface
- Includes an internal SGML parser that ensures documents are fully ISO 8879 (SGML) compliant by validating the markup.

- Provides formatted and unformatted views of the document.
  The formatted view gives the writer a good idea of what the document will look like when it is printed.
  The unformatted view allows the writer to see the text and the markup (elements or tags).
  These views and the parser help ensure that the document will be fully compliant.
- Includes the CALS Document Type Definition and Document Type Definition according to ISO Technical Report 9573, as well as Document Type Definitions for DCF GML Starter Set and DCF SampleDoc applications as examples for publications departments to create their own non-CALS Document Type Definitions.
- Editor will support SGML features.
- Includes spell-checking.
- A graphic may be referenced in an SGML document and viewed in a separate window with an appropriate graphic editor.

C.14.2 SGML Text Write Tools OS/2 Edition Product Overview

SGML Text Write Tools OS/2 Edition is a software program that helps users:

- Create new Document Type Definitions
- Modify existing ones
- Write screen formatting in support of a particular Document Type Definition.

SGML Text Write Tools OS/2 Edition will run in an OS/2 environment under Presentation Manager and will exploit the inherent features in the windowing environment. The product enables the user to:

- Modify the parameters associated with formatting of the Document Type Definition components for a local display and printing. These parameters include font, point size, color, indentation, generated text, and automatic numbering,
- Create and/or modify an application that is similar in content to an existing application.

SGML Text Write Tools OS/2 Edition requires SGML Text Write OS/2 Edition to be installed on the workstation as a prerequisite.

C.15 IBM Enterprise Printing Overview

This section contains redundant information. Particularly, this section introduces again, in a short form, the products described in enterprise-wide printing.

For more detailed information on the different products, refer to documents referenced in Appendix E, "Bibliography" on page 229.

C.15.1 Products Provided for Enterprise Printing

- IBM 3900 Advanced Function Printer
- Advanced Function Image and Graphics Feature
- IBM LaserPrinter 4028 Model NS1
- PSF/MVS, PSF/VM, and PSF/VSE Version 2
- IBM AFP Core Interchange Fonts

- Document Composition Facility (DCF) Release 4
- Overlay Generation Language/370 and Page Printer Formatting Aid/370
- IBM SAA PrintManager/400 in OS/400 Release 3
- IBM SAA PrintManager for MVS and VM
- Further hardware and software

Highlights IBM's Advanced Function Printing system provides:

- Support for a wide range of IBM printers, including desktop, high-speed, cutsheet and fanfold printers
- Local and remote attachment
- Fast, efficient handling of image and graphics
- Support for printer-resident bar code processing
- Fonts that allow data to be printed and displayed with consistency across environments
- Flexible electronic form overlays
- Automated data security, integrity, and auditability
- An architected platform for easily integrating new printers and printing function
- An open, published architecture that is part of SAA
- Application independence from specific printer characteristics or attachments.

Relationship to system/390 The Advanced Function printers along with currently supported page printers, are supported on both System/370 and the System/390 processors.

The Advanced Function Printing software products are supported in their intended operating environments by the current levels of the respective operating systems.

C.15.2 Description

IBM Advanced Function Printing (AFP) and SAA IBM Systems Application Architecture* (SAA) is the framework for developing consistent applications across offerings of the major IBM computing environments. The architectural element of SAA Common Communication Support (CCS) for the creation of compound documents (documents that contain text, image and graphics) is Mixed Object Document Content Architecture for Presentation MO:DCA-P). The AFP data stream (AFPDS) was published in September 1989, as the SAA compliant print data stream for applications. MO:DCA-P is supported as a subset of the AFP data stream. System programs convert the AFP data stream into the SAA Intelligent Printer Data Stream (IPDS) for the targeted printer.

Also, the IBM SAA PrintManager product, available for MVS, VM, and provided as part of the operating system of OS/400, provides an SAA interface for enterprise printing providing consistency, management, and portability of print applications.

IBM Advanced Function Printing Products

The heart of the system is the Print Services Facility (PSF), which converts AFPDS or line data applications into the IPDS printer stream. Differences between most printer characteristics and carrier protocols, such as channels and telecommunication lines, are handled automatically by PSF.

PSF also provides system management functions such as automatic resource management, spool management for error recovery, checkpoint restart and workload balancing, message routing, and print job accounting.

Please refer also to Chapter 12, "Print and View within the Different Environments" on page 77 for a more complete overview of the printer driver in the different environments.

In the OS/400 environment, a Print Services Facility-like function is integrated into the base operating system.

The IBM SAA PrintManager allows programmers to select and validate printers and print options from within an application program in a consistent way in the MVS, VM and OS/400 environments without the use of system specific commands.

A set of core interchange fonts, including two typographic families and a family of fixed-space fonts, enhance document interchange between displays and printers in the OS/2 environment and printers attached to S/370* systems. The IBM enterprise print products support a family of 14 cutsheet and fanfold IBM printers in a range of speeds from approximately 3 to 229 impressions per minute (ipm).

C.15.3 AFP Printers, Printer Features, and Attachments

IBM 3900 Advanced Function Printer The IBM 3900 Advanced Function Printer is a 229 ipm continuous forms page printer. It is based on the same control unit as the 3825, 3827, and 3835, and is a fully compatible member of the AFP printer family.

IBM 3825, 3827, 3835 and 3900 AFIG Feature The Advanced Function Image and Graphics feature allows processing of compressed image and vector graphics orders on the 3825, 3827, 3835 and 3900 Page Printers. Applications that produce the IBM SAA Image Object Content Architecture (IOCA) or Graphic Object Content Architecture (GOCA) formats may now be printed on a high-speed system printer.

IBM Laserprinter 4028 The IBM LaserPrinter 4028 is a cutsheet, simplex laser printer with print speeds of up to 10 ipm and attaches to the 3270 family of controllers and adapters. It is based on the same print engine as the IBM 4019 LaserPrinter with 300 picture element (PEL) addressability.

Print Services Facilities PSF/MVS and PSF/VM
These versions provide common print support in MVS and VM environments for enabling print applications on the System/370 SAA platform.

Version 1 supports most of the current available AFP printers. In addition Version 2 supports the printer 4028 and provides improved ways to support printing in comparison to version 1.

Version 2 of PSF provides a simplified packaging method which incorporates the key printer attachment features into the PSF base product.

Version 2 of PSF in the System/370 environments supports the following printers and printer features:

- 3900 Advanced Function Printer
- Advanced Function Image and Graphics feature
- IBM LaserPrinter 4028
- IBM 4224 and 4234 workstation printers
- Printer-based bar code processing

Please refer also to Chapter 12, "Print and View within the Different Environments" on page 77 for a more complete overview of the printer driver in the different environments.

PSF/VSE Version 2 PSF/VSE Version 2 supports:

- 3900 Advanced Function Printer
- 3812 and 3816 Printers
- Remote PrintManager for the IBM 3820, 3825, 3827 and 3835 Printers
- Conditional processing for line data applications, like that in PSF/MVS and PSF/VM
- Compressed image

PSF/VSE Version 2 also provides support for POWER JCL commands, job restart capability, and a sample utility for transferring print resources from an MVS or VM system to VSE.

New IBM AFP Core Interchange Fonts New IBM AFP Core Interchange fonts are available with Version 2 of PSF/MVS, PSF/VM and PSF/VSE. These fonts will provide consistent printing across AFP platforms and printing of OS/2 Presentation Manager based applications on host attached printers. Availability of display and printer fonts with the same typeface and metrics enables Presentation Manager applications to provide WYSIWYG (what you see is what you get) capability on printers.

The Core Interchange fonts include three general Latin language type families:

Times New Roman
Helvetica
Courier

These fonts match in typeface and metrics those available in the OS/2 environment for display on the PS/2. They also include non-Latin language fonts.

C.15.4 Print Application Development Aids

C.15.4.1 Overlay Generation Language/370 and Page Printer Formatting Aid/370

IBM's host-based batch products for creating print formatting resources, Overlay Generation Language (OGL) and Page Printer Formatting Aid (PPFA), have been replaced by new cross system programs that provide enhanced functions in all supported System/370 environments: MVS, VM and VSE.

Please refer also to C.7, "Page Printer Formatting Aid/370" on page 187 for a more complete overview of the tools provided to generate print resources.

C.15.4.2 IBM SAA PrintManager

IBM SAA PrintManager for MVS and VM implements the PrintManager interface allowing application output to be placed on system spool for printing in a consistent manner. It provides a Print Request Facility (PRF) for directing output to any printer in the enterprise, and enables automatic printing of PS/2 application output on host-attached printers, via the IBM SAA Connection Services/MVS and VM.

Application programs that are written to the IBM SAA PrintManager interface and use the PrintManager print descriptors can use a common set of print options from within an application program. Print files may then be sent to the system spool in a simplified, consistent manner. These SAA-compliant applications may be moved between VM, MVS, and OS/400 environments with little or no modification. Users may submit jobs for printing on any printer, including AFP printers. Logical printer names are used for selecting and validating print options before placing a print job on the system spool, reducing the possibility of print errors.

C.15.4.3 IBM SAA PrintManager/400

IBM SAA PrintManager/400 is the AS/400 implementation of the PrintManager interface, the print element of the SAA CPI. Applications developed using PrintManager/400 in the OS/400 environment (or PrintManager in MVS or VM) may be moved to any supported environment without changing the way the application selects printers and print options, or places print data on the system spool.

C.15.5 Integration with Applications and Application Environments

C.15.5.1 Document Composition Facility (DCF)

DCF provides document formatting function for IBM host-based publishing. Release 4 functions include:

- Language improvements, including hyphenation and index sort support for six additional languages (Portuguese, Swedish, Finnish, Icelandic, Norwegian and Danish); and algorithmic hyphenation for all 15 DCF languages
- Shading for boxes, areas, and tables for AFP and PostScript devices
- The ability to produce separation masters for multiple color printing and multi-part forms
- Support for including page overlays with variable origin positions (floating overlays)
- Page segment handling enhancements, including compressed image in page segments, an ABSOLUTE parameter on the .SI control word, and adding the library name in error messages
- Logical device support for the new IBM LaserPrinter 4028 300 dpi printer

There are now online HELP screens for messages and enhancements to revision codes.

DCF includes the optional SCRIPT Mathematical Formula Formatter feature for supporting documentation for science and engineering applications.

C.15.5.2 LOTUS 1-2-3/M

LOTUS 1-2-3/M is available as a host-based component of a 1-2-3 based Enterprise Spreadsheet System which supports mainframe line and all-points-addressable (APA) printers. Lotus 1-2-3/M produces AFPDS files for APA printing in addition to files with ANSI line printer controls and files without printer controls.

C.15.5.3 WordPerfect/370

WordPerfect available on VM/SP/XA (CMS Version 4 Release 12 or higher) operating system, supports AFP by generating the AFPDS data stream. WordPerfect has indicated intent to provide similar support for TSO/E on the MVS operating system.

C.15.5.4 IBM Publishing Systems BookMaster

Publishing Systems BookMaster, IBM's host-based publishing solution, provides a wide range of services for creating, formatting, and managing documents, including text, image, and graphics. BookMaster output is based upon the DCF text formatter.

C.15.5.5 IBM BookManager

The IBM BookManager solution is an SAA application that allows the provision of softcopy documentation to users anywhere in the enterprise, from a centrally managed online library. IBM product documentation may be customized or combined with customer documentation to provide printed or online manuals.

C.15.5.6 IBM GDDM

GDDM provides support for AFP applications. GDDM and its associated programs may be used for creating business graphics and images for printing under AFP as documents or page segments.

When using GDDM with IBM BookMaster, and publishing functions in general, ensure that current levels of PTF are always applied; the publishing image and graphics world is fast changing.

C.15.5.7 IMS/ESA V.3. R.2.

IMS/ESA* includes support that allows IMS applications to send print output to the JES2 spool via an IMS Spool API. This capability, similar to that provided for CICS applications in CICS/MVS, allows IMS applications to take full advantage of AFP program and printer capabilities.

C.16 IBM SAA PrintManager

IBM SAA PrintManager provides common access to printing, including Advanced Function Printing (AFP) across the supported Systems Application Architecture (SAA) environments: VM, MVS, and AS/400

Through the use of the PrintManager Interface and the application programming interface (API), a common set of print options from within an application program can be employed.

Print files can then be sent to the system spool in a simplified, consistent manner. These applications can be moved between SAA platforms with little or no modification. Additional ease of use and function are provided by the PrintManager Print Request Facility (PRF).

The Print Request Facility provides a user with a consistent way to submit print jobs. The PRF provides both a command interface and a set of interactive menus, which are ISPF based. These menus can be customized to meet the needs of an enterprise.

Printing concept The sequence of printing is (please refer to the following figure):

1. An end user selects a printer by its printer descriptor name.
2. The end user is allowed to override predefined print options or specify additional options.
3. PrintManager put the job on the system spool, from which a printer driver can select it for printing.

- When printing using PrintManager in distributed systems, a communication program sends the job to the spool of the receiving system and printing can then occur.

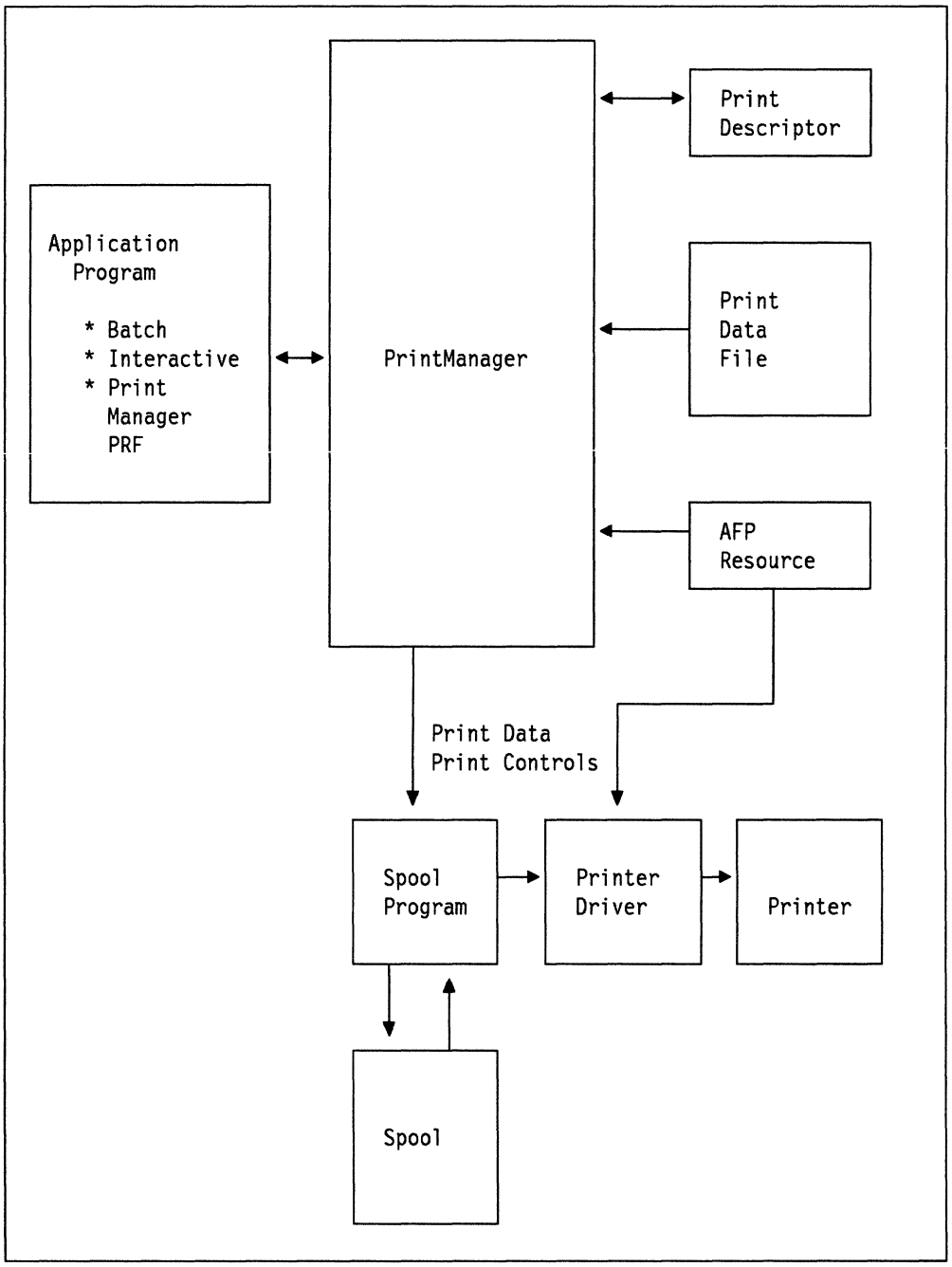


Figure 35. SAA PrintManager Model. This figure shows an overview of the general concepts of printing using SAA PrintManager.

C.17 Advanced Function Image and Graphics Feature and Decompression Performance Enhancement Feature

The Advanced Function Image and Graphics Feature supports the printing process by decompressing images and by preparing graphical data for printing in the printer.

This feature is available for the following printers:

- IBM 3825

- IBM 3827
- IBM 3835
- IBM 3900

In addition, other channel attached printers can have the Decompression Performance Enhancement (DPE) feature.

Description of the AFIG feature The AFIG feature provides several functions. It improves IM1 image performance intensively. It supports printer microcode image decompression of Image Object Content Architecture (IOCA) data streams that were compressed by industry standard compression routines:

- CCITT Group 3 and 4
- Modified Modified Read (MMR)
- Adaptive Bilevel Image Compression (ABIC)

IBM Advanced Function Image and Graphics support is able to correct the resolution of an input image to match the printer's 240 pel resolution and thereby provide input resolution independence for scanned input. Image scaling is supported to expand or contract images in the printer. Scale to fit, center and trim, and position and trim are some of the additional image mapping functions supported in the printer. Images can be clipped in the printer as well as rotated 90, 180 or 270 degrees.

Graphics Object Content Architecture (GOCA) graphics commands are supported for execution by the printer control unit.

Description of the DPE feature The DPE feature supports printer hardware image decompression of Image Object Content Architecture (IOCA) data streams that were compressed using standard compression routines:

- CCITT Group 3 and 4
- Modified Modified Read (MMR)

The DPE capability improves again the decompression performance offered with the AFIG feature.
The AFIG feature is a prerequisite for the DPE feature.

C.18 Publishing Systems BookMaster

- BookMaster provides a rich set of tags for the creation and layout of documents that can be composed and delivered via printed output or for online viewing using BookManager.
- The user can specify the character set for the specific language he or she requires using Country Extended Code Pages.
- BookMaster incorporates functions for the definition and creation of separation masters for multi-color printing.

Business Solutions

- BookMaster, DCF, and the BookManager BUILD and READ family of products provide a host-based solution for professional and enterprise publishing. BookMaster helps generate printed output and online softcopy books that can be displayed and viewed on

host terminals as well as IBM Personal System/2s and personal computers under several operating systems.

- BookMaster can be used with the IBM Publishing Systems ProcessMaster (VM and MVS Edition) products. In conjunction with the other Master Series and BookManager products, ProcessMaster provides facilities for source file management, transformation of data formats, and document preview prior to printing.

C.19 IBM Publishing Systems TextTagger

IBM Publishing Systems TextTagger provides a way for data generated on PC-based word processors to be integrated into BookMaster or SGML data bases. This provides a pathway for PC generated text to be incorporated in BookMaster or SGML documents or BookManager books.

The programmability of IBM Publishing Systems TextTagger provides a way to create custom tagging applications that convert data from PC formats to host formats.

Productivity of authors using word processors is enhanced by providing a tool that can minimize the amount of work required to convert documents from:

- ASCII printer files
- Plain text files (both ASCII and EBCDIC)
- PDA (Processed Document Architecture) files
- Revisable-Form Text Document Content Architecture (RFTDCA) files
- WordPerfect 5.0 and 5.1 files
- LIST1403 files

to formats such as:

- BookMaster
- CALS
- SGML Starter Set

This allows professional writers to access data generated from other sources without having to re-key and re-tag the data.

C.20 Publishing Systems BrowseMaster

BrowseMaster assists departments in electronically publishing documents by importing and converting graphical illustrations, image information, Computer-Aided Drafting (CAD) plot information, and then previewing formatted data before sending it to a supported printer.

BrowseMaster offers several import and conversion facilities.

CALS user's can convert the CALS Computer Graphics Metafile (CGM) subset into ADMGDF format for viewing and cropping. CGM can also be produced from ADMGDF. CALS-conforming image information CCITT G4 compression can be converted into page segments for CALS publications.

Other bilevel image formats can be converted back and forth from the CALS format, page segment, Image Object Content Architecture (IOCA), and others.

Likewise, CAD information, California Computers Products Incorporated (CALCOMP**), Graphics Language (GL), and ADMGDF) can be converted into page segments for output. These files can also be converted to ADMGDF first (for viewing and cropping) before conversion to page segment.

Note: BrowseMaster is a subset of Graphical Display and Query Facility (GDQF). A customer could start with BrowseMaster and as additional function is needed, migrate to the GDQF Base (5688-169) product.

C.21 Graphical Display and Query Facility

With GDQF, publishing departments can electronically combine graphical illustrations with text, preview formatted data before sending it to print, and convert data to formats used in publications. GDQF offers a set of utilities to quickly and easily convert several data types for inclusion in documentation. Computer-Aided Acquisition and Logistics Support (CALS) users can view, edit, and convert image information into CCITT G4 (CALS image format) or printer graphics formats for CALS publications.

- Text Text in multiple type sizes and type styles.
- Vector graphics data bar and pie charts, line art, engineering drawings, and flow diagrams.
- Image data scanned or faxed data.
- Raster image data scanned data that is stored in the system as a series of bit maps or byte maps (black and white or shades of gray).

Office productivity is improved by providing the ability to send drawings to non-engineers for review. Executives can view data, monitor changes, annotate, and return data to the originator. Communication between engineering and office is enhanced by providing office personnel access to graphical data. Graphics can be faxed, edited, and stored in the data repository. Graphics can also be attached to electronic mail and stored in the office database. GDQF also connects to the IBM Professional Office System (PROFS*), providing information access. Marketing departments can use GDQF by accessing drawings for sales information. Sales trends can be effectively analyzed by creating business graphics with the Interactive Chart Utility available under GDQF. Business graphics can also be converted to formats used for publishing technical documents.

Since GDQF is a very powerful tool in converting from one data type to another, please refer also to 14.1, "Format Conversions" on page 125 and 14.2, "Image Conversion under GDQF" on page 127.

C.22 Image Handling Facility

The focus of Image Handling Facility, Version 2 is the preparation of pictures (images) to be included in documents. It supports the interactive manipulation of images in a manner similar to the photographic reproduction process. Images can be line art (drawings) or continuous tone pictures (photographs). They may be monochrome (black/white) as well as color.

Highlights

- Rotate and shear through any angle
- Gray level image display

- Crop, invert, and requantify gray level images
- Color conversion for gray level images
- TIFF support
- CCITT Group 4 import without header
- PostScript output
- Multi-line text input (GDDM Vector Symbol Set or 4250 Fonts)
- Interactive 4250 Font selection
- Compression/Decompression enhancement
- Support of variable image widths

C.23 Graphical Data Display Manager (GDDM)

The IBM Graphic Data Display Manager (GDDM) is the base for many graphic applications in different IBM environments.

C.23.1 Products Included:

- GDDM/VM
- GDDM/VMXA
- GDDM/MVS
- GDDM/VSE
- GDDM-PGF (Presentation Graphics Facility)
- GDDM Interactive Map Definition
- GDDM-IVU (Image View Utility)
- GDDM-REXX
- GDDM-GKS (Graphical Kernel System)
- GDDM-CSPF (Central Slide and Plot Facility)
- GDDM-OS/2 Link
- GDDM-PCLK

The Graphical Data Display Manager (GDDM) Series consists of the licensed programs listed above. GDDM is IBM's primary device support and host graphics program, supporting graphics and images on displays, 3270-PC family, 5080 Graphics System, 3117 and 3118 Scanners, and printers.

Application programs call GDDM routines to do full-screen alphanumerics or images, or to create business charts. Applications may also use graphic input and display to perform functions with information such as cartographic data that is meaningful only when a pictorial format is used.

The interactive chart utility (ICU) of GDDM-PGF allows users to create and customize business charts and graphs without the need to write application programs.

All products under GDDM are briefly described below.

GDDM/VM, GDDM/VMXA, GDDM/MVS, and GDDM/VSE These programs provide a set of user-callable subroutines that provide graphic and image functions and management of alphanumeric fields for terminal display stations, printers, and plotters.

GDDM generates the data necessary to make up the display/printer picture, manages the complex data stream required for transmission, and minimizes the data-stream lengths through compression.

GDDM subroutines provide:

- Screen format control
- Alphameric display and input
- Graphics construction and display (lines, arcs, text)
- Control of attributes
  (color, line type, line width, symbols, shading patterns)
- Control of graphic input
- Control of display partitions
- Display and printing control
- Printing and viewing of composite document files containing formatted text, graphics, and images

Image Symbol Editor GDDM also contains the image symbol editor (ISE) utility, which permits the user to interactively define, edit, and save sets of images.
ISE can create programmed symbol sets that can be saved by the utility and later loaded by GDDM or another program that provides the symbol set load capability. ISE can also create patterns, markers, and larger image symbols that can be used by GDDM applications.

GDDM-PGF PGF contains a comprehensive set of high-level presentation graphics and charting callable routines oriented toward business graphics. They provide support of the following functions:

- Chart types:
  - Bar charts (both vertical and horizontal)
  - Tower charts
  - Histograms
  - Line graphs
  - Scatter plots
  - Pie charts
  - Surface charts
  - Venn diagrams
  - Polar charts
  - Table charts
  - Graphics-text-only charts
- Functions that can be specified:
  - Position and size of the chart
  - Headings
  - Axes
  - Lines and grids
  - Key symbols to identify chart components
  - Attributes (line types, marker symbols, color shading patterns, and symbol sets)
  - Ten selectable type styles
  - Degree of curve smoothing
  - Chart annotation

GDDM-PGF also includes two utilities:

- **The interactive chart utility (ICU)** enables a user either to create business graphs and charts without application programming or to call the utility from an application program. Assisted by menus that are presented on the display, the user can:
  - Enter data
  - Create and customize the same types of charts and graphs that can be created by the callable subroutines
  - Save, for subsequent use, information about the chart format and data
  - Print charts
- **The vector symbol editor** provides the user with the ability to construct vector symbol sets for use by GDDM. Symbols have the following characteristics:
  - They are defined by a series of lines or curves drawn from point to point in a defined space.
  - They can be used by calls to GDDM from an application program or from the interactive chart utility.
  - They can be scaled to desired size, sheared, colored, and drawn at an angle/direction by calls to GDDM.

GDDM Interactive Map Definition GDDM Interactive Map Definition enables interactive definition of alphanumeric screen and printer layouts known as maps. The maps can be used in conjunction with GDDM application programs and can contain a graphics field (but not an image field).

GDDM Image View Utility GDDM-IVU adds ease of use to the GDDM base image application programming interface (API) by providing higher-level processes for handling images. GDDM-IVU can be accessed interactively by end users or by user-written application programs. The image processes facilitate user tasks such as:

- Input: Import/scan/load
- Process: Projection definition/image manipulation/viewing
- Output: Export/print/file

GDDM-IVU is used for transforming data types. Please refer to 14.1, "Format Conversions" on page 125 to see where it is of help.

GDDM/VMXA GDDM/VMXA offers functions similar to GDDM/VM and in addition exploits the capabilities of the VM/XA* operating system, in particular, 31-bit addressing and virtual machines larger than 16 megabytes. GDDM/VM is functionally stabilized at Version 2 Release 2 and is superseded by GDDM/VMXA for VM/SP Release 6 and VM/IS 6.

GDDM-REXX GDDM-REXX allows almost any GDDM/VM, GDDM/VMXA, GDDM/MVS, GDDM/VSE, GDDM-PGF, GDDM-IVU, and GDDM-GKS call to be coded in a REXX-language procedure running under CMS.

GDDM-Graphical Kernel System GDDM-GKS provides a graphics application programming interface under TSO and CMS that is an alternative to the GDDM base products. It is an implementation of the International Graphical Kernel System standard, ISO 7942.

GDDM-Central Slide and Plot Facility GDDM-CSPF is a software package for post-processing GDDM graphics. Starting with print files created by GDDM applications such as the interactive chart facility (ICU), it produces high-quality 35-mm slides and prints, overhead transparencies (foils), and paper plots.

GDDM-PCLK Users of PC DOS on personal systems can use GDDM-PCLK to run GDDM graphics application programs in the host under CMS, TSO, and CICS. A personal system with a graphics display adapter and 3270 terminal emulator can access a set of end user functions similar to those of a 3192 terminal, including color graphics and alphanumerics, a mouse, and an attached printer or plotter. GDDM pictures can be saved in PC files for later processing. PCLKF (PCLK feature) is required on the GDDM base programs in the host if GDDM-PCLK is to be used in the personal system.

GDDM-OS/2 Link Users of OS/2 EE Version 1.2 or later can use GDDM-OS/2 Link to run GDDM applications in the host under CMS, TSO, and CICS. GDDM-OS/2 Link makes use of the OS/2 Presentation Manager functions, including windowing, print, plot, and file interchange to PM Metafile. GDDM-OS/2 Link is for users who want the type of function of GDDM-PCLK Version 1.1 under OS/2.

C.24 ADMUCIMV

Here is the VM version of this important EXEC. It can be used for several functions, but not all are fully explained in the help given with the EXEC, but a brief inspection will show how to select the appropriate values of parameters. Please refer to the appropriate places and check that the procedure has the options set correctly for you.

Ensure the correct definitions are set for:

- Device Token
- Document Type
- Data Stream Type
- Image Format

Other values may be specified when calling this procedures, or the default values taken.

```
* NAME:     ADMUCIMV
*
* FUNCTION: DRIVE MODULE ADMUCDSO TO OUTPUT CHARTS OR GDF
*           FOR A COMPOSED PAGE   (FAMILY-4) PRINTER
*
* INPUT:    CHART_DATA_NAME  CHART_FORMAT_NAME|'GDF'
*
*                5668-812
*                (C) COPYRIGHT IBM CORP. 1979,1986
*                LICENSED MATERIALS - PROPERTY OF IBM
*
*
&CONTROL ERROR
*
* REQUEST INPUT PARAMETERS IF NOT PRESENT ON ENTRY
*
&IF &INDEX > 0 &GOTO -PARMOK
&SPACE
&BEGTYPE
==> ENTER FILENAME OF CHART DATA (AND FORMAT IF DIFFERENT):
&END
&READ ARGS
&IF &INDEX = 0 &EXIT 4
*
* PROCESS CHART DATA NAME
```

```
*         CHART FORMAT NAME (OPTIONAL)
*
-PARMOK
&CD = &1
&CF = =
&IF &INDEX = 1 &TYPE *** CHART FORMAT NAME ASSUMED = CHART DATA
NAME.
&IF &INDEX > 1 &CF = &2
*
* ICU DISPLAY OPTION:
*             DEFAULT DISPLAY = 6
*             IF THE DATA FILE IS GDF THEN DISPLAY = 99
*             THIS CAUSES ADMUCDSO TO LOAD THE GDF AND DISPLAY
*             DIRECTLY ONTO THE REQUESTED DEVICE.
*             THIS IS NOT A CHART DISPLAY OPTION.
*
&DP = 6
&IF &CF = GDF &DP = 99
*
* DEFINE DEVICE TOKEN: (IMG600X) 4250 PRINTER
*                      (IMG240X) 3800-3/3820 PRINTER
*
&DT = IMG600X
*
* DSOPEN OPTION GROUP 5
* =====================
* DEFINE DATA STREAM TYPE: (0) DOCUMENT (PRIMARY D/STREAM)
*
&DS = 0
*
* DSOPEN OPTION GROUP 6
* =====================
* DEFINE SPILL FILE USAGE: (0) KEEP INTERNAL DATA IN A SPILL FILE
*                          (1) KEEP INTERNAL DATA IN MAIN STORAGE
*
&SP = 0
*
* DSOPEN OPTION GROUP 7
* =====================
* DEFINE THE NUMBER OF SWATHES THAT MAKE UP THE COMPLETE IMAGE
*
&N = 8
*
* DSOPEN OPTION GROUP 8
* =====================
* DEFINE IMAGE WIDTH
*             DEPTH
*             UNITS: (0) TENTHS OF AN INCH
*                    (1) MILLIMETERS
*
&W = 60
&D = 40
&U = 0
*
* DSOPEN OPTION GROUP 9
* =====================
* DEFINE IMAGE FORMAT: (0) UNFORMATTED (BIT ARRAY)
*                      (1) FORMATTED D/STREAM TO CDPF/PSF REQUIREMENTS:
*                          (CDPF IS THE 4250 DEVICE DRIVER)
*                          (PSF IS THE 3800-3/3820 DEVICE DRIVER)
*
&FO = 1
*
* DSOPEN NAMELIST PARAMETERS
* ==========================
* DEFINE OUTPUT IMAGE FILE_NAME
*                     FILE_TYPE
*                     FILE_MODE
*
&F = &CD
&T = *
&M = A1
*
* PRINTOUT OF ALL VARIABLES FOR A VISUAL CHECK
*
```

```
&SPACE
&TYPE . CHART DATA NAME = &CD
&TYPE CHART FORMAT NAME = &CF
&TYPE .... DEVICE TOKEN = &DT
&IF &DS = 0 &IF &FO = 1 &TYPE .DATA STREAM TYPE = DOCUMENT
&IF &DS = 1 &IF &FO = 1 &TYPE .DATA STREAM TYPE = PAGE
SEGMENT
&IF &SP = 0 &TYPE .SPILL FILE USAGE = REQUIRED
&IF &SP = 1 &TYPE .SPILL FILE USAGE = NOT REQUIRED
&TYPE NUMBER OF SWATHES = &N
&TYPE ..... IMAGE WIDTH = &W
&TYPE ..... IMAGE DEPTH = &D
&IF &U = 0 &TYPE .IMAGE SIZE UNITS = TENTHS OF AN INCH
&IF &U = 1 &TYPE .IMAGE SIZE UNITS = MILLIMETERS
&IF &FO = 0 &TYPE .... IMAGE FORMAT = BIT ARRAY
&IF &FO = 1 &IF &DT = IMG600X &TYPE .... IMAGE FORMAT = CDPF
D/STREAM
&IF &FO = 1 &IF &DT = IMG240X &TYPE .... IMAGE FORMAT = PSF
D/STREAM
&TYPE . OUTPUT FILENAME = &F
&TYPE . OUTPUT FILETYPE = &T
&TYPE . OUTPUT FILEMODE = &M
*
* GO/QUIT DECISION
*
&SPACE
&TYPE ==> ENTER GO OR QUIT:
&READ VARS &REPLY
&IF .&REPLY NE .GO &EXIT 100
*
* PARAMETERS AT ADMUCDSO ENTRY:
*
*   CHARTDATA  CHARTFORM  CHARTDISP  DSOPEN-PARAMETERS
*
*   CHARTDISP IS THE DISPLAY PARAMETER FOR CSSICU.
*
*   THE DEVICE-ID IS OMITTED FROM DSOPEN-PARAMETERS. THE
*   PROCLIST AND NAMELIST ARE WRITTEN IN BRACKETS SO THAT THE
*   PROCOPT-COUNT AND NAMELIST COUNT ARE DEDUCED BY ADMUCDSO
*
* PARAMETERS AT ADMUCDSO DSOPEN CALL TO GDDM:
*
*    FAMILY  DEVICE-TOKEN  ( PROCOPT-LIST )  ( NAMELIST )
*
ADMUCDSO &CD &CF &DP 4 &DT (5 &DS 6 &SP 7 &N 8 &W &D &U 9 &FO)(&F &T &M)
*
&EXIT &RC
```

C.25 BookManager Family Overview

Why BookManager? Information is a vital asset, and much of that information is contained in the manuals, publications, proposals, and other structured documents typically published in paper, hard copy, form. This information is often voluminous, complex, and needs frequent updating. Libraries of these documents require extensive storage space and manual management systems. Finding the information needed to perform required procedures, answer customer inquiries, or complete an analysis can be time consuming and error prone.

What is BookManager? BookManager's softcopy application enabling tools provide an efficient means to put information online and to use it with improved productivity. Information that can also be used to produce printed documents is converted to an online, or softcopy form that can be viewed, searched, annotated, copied, and printed at end users' workstations in a number of key IBM systems environments.

BookManager consists of a group of programs available on the mainframe and on workstations:

- BookManager READ (VM, MVS)
- BookManager BUILD (VM, MVS)
- BookManager Read/DOS
- BookManager READ/2

Highlights

- Helps organizations manage published information by enabling computerized online delivery of information traditionally provided in hard copy
- Enables enterprise and cross-enterprise documentation applications by providing:
  - A common online information format
  - BookManager programs to provide softcopy functions across platforms
- Enables IBM to ship softcopy versions of IBM product manuals to customers and meet their requirements for electronic distribution and storage
- Allows companies using IBM DCF and Master Series products or SGML to move from hard-copy to softcopy publishing using existing source files and without significant special effort
- Helps end users find information faster and handle larger libraries by providing new information retrieval techniques
- Improves end user productivity by enabling online annotation, copying and printing of information retrieved

Note: When moving files between host and workstation, remember to use binary file transfer, and when uploading to the host, to specify record format F and record length 4096.

```
RECEIVE filename.BOO  filename BOOK
SEND    filename.BOO  filename BOOK ( RECFM F LRECL 4096
```

C.26 Publishing Systems PostScript Interpreter for AFP

Publishing Systems PostScript Interpreter transforms PostScript files into AFPDS format for printing on the AFP printer family.

This capability provides data interchange between host and workstation-based publishing systems and allows the use of IBM printers and networks for documents created on PostScript language-based publishing systems.

The PostScript Interpreter for AFP processes files with PS (PostScript), EPS (Encapsulated PostScript), and LISTPS filetypes to create files that users can print on several IBM printers. The type of output the Interpreter creates depends on the release of the program.

This program is available under MVS and VM. If it is installed under VM, just enter the command `PS370`.

Highlights

- Transformation of PostScript files for printing on IBM AFP printers. Convert either to page segment (PSEG38xx) or to a document (LIST38xx)

- Interchange between host and workstation-based publishing systems
- Allows use of AFP networks for printing PostScript documents
- Provides means to install type faces
- Provides means to build a 3820 font from PostScript fonts

C.27 Ventura Publisher

Ventura Publisher is a widely used desk-top publishing program.

It is notable in that it stores text in a humanly readable form with tags at the beginning of document elements, which though essentially format-related (they are effectively macro names which define specific formatting) can in many circumstances be treated as if they were intent based markup similar to that defined in GML and SGML systems; Ventura Publisher, and programs like it, can be used with a little ingenuity as a bridge between the desk-top world and that of GML.

Ventura tags are largely defined by the user, and formatting values associated with them. For example, we have seen `@PARA =` as a paragraph tag, and `@BULL =` for a bulleted item in a list. It is not a long step to change these to :p. and :li.

Ventura Publisher does also have direct formatting controls; but again these are clearly distinguished, and can often be simply converted. For example, as long as the creator of a document has been reasonably consistent, and has used simple controls, the highlighting in a document may consist simply of `<B>` (Bold) and `<BI>` (Bold Italic) format controls. (These controls are automatically turned off at the end of a paragraph or with a `<D>` control.) Again, it doesn't take a complex filter to change these to `:hp1.`, `:hp2.`, and their associated end tags.

C.27.1.1 Formatting Codes

Table 14 (Page 1 of 2). Some Ventura Publisher Formatting Codes

Code	Meaning
<B>	Bold type
<I>	Italic type
<M>	Medium weight type
<S>	Small type
<^>	Superscript
<V>	Subscript
<U>	Underline
<O>	Overscore
<=>	Double underline
<Pn>	Change font size to point size *n*
<N>	Space where line break not allowed
<R>	Newline
<$!text>	"text" is a non-printing comment
<$Ftext>	"texet" is placed in a footnote referenced from this position

Table 14 (Page 2 of 2). Some Ventura Publisher Formatting Codes	
Code	**Meaning**
<$n/m>	Prints fraction *n* over *m*
<$n over m>	Prints fraction *n* over *m*

C.28 IBM Interleaf Publisher

The IBM Interleaf Publisher program provides the user with functions to create, edit and manage documents containing textual, graphical, and image data.

- It provides a fully integrated desktop publishing environment. The user is able to perform text editing, graphics editing and image editing within a uniform set of operating procedures including keyboard commands and pop-up and pull-down menus.
- Provides native PostScript to a PostScript printer or to an output data file.
- Provides plain ASCII output from the text components of IBM Interleaf Publisher documents.
- Supports incorporation of the following textual and graphical data formats within the limitations of the IBM Interleaf Publisher:
  - ASCII text
  - IBM RFT DCA
  - MicroSoft Word RTF format
  - Multi-Mate
  - WordPerfect files in IBM RFT DCA format
  - IBM 3117/3118/3119 scanner image data stream (IMDS) (bi-level or gray-scale).
  - Tagged image format files (TIFF) (bi-level or gray-scale, uncompressed or compressed).
  - Encapsulated PostScript files
  - Hewlett-Packard Graphics Language (HP-GL) files
  - Other complementary data formats
- Provides text editing functions
- Provides graphics functions
- Free hand drawing
- Provides image functions
- Capability to share documents and fonts with other users attached to a common network file server.

C.29 IBM ImagePlus Workstation Program Family

IBM ImagePlus Workstation Program Family is part of the IBM SAA ImagePlus Folder Application Facility MVS/ESA with IBM SAA ImagePlus Object Distribution Manager MVS/ESA, and the IBM SAA ImagePlus Workfolder Application Facility/400 environments.

IBM has designed the ImagePlus Workstation Program Family based on the following:

- IBM ImagePlus Workstation Program upgradability gives the customer a consistent upward migration path as workstation technology evolves.
- Use of standard architectures provides a consistent, standardized environment within which system growth may take place.

- IBM Object Content Architectures:
  - MO:DCA-P (Mixed Object: Document Content Architecture - Presentation) - used as an envelope to contain the document to handle multi-page image documents.
  - PTOCA (Presentation Text Object Content Architecture) - used to describe a page of coded data.
  - IOCA (Image Object Content Architecture) - used to describe a page of image data.
- Image Data Compression: IBM Modified Modified Read (MMR); Consultative Committee on International Telephone and Telegraph (CCITT) Group 4; Adaptive Bi-level Image Compression (ABIC) bi-level compression techniques; Joint Photographic Expert Group (JPEG) color compression.

- Common User Access (SAA/CUA) compliance provides the IBM SAA ImagePlus Workstation Program/2 user with a familiar workstation environment, one consistent throughout all SAA-compliant systems.
- Common Communications Support (SAA/CCS) compliance provides the IBM SAA ImagePlus Workstation Program/2 user with a consistent communications environment, enabling distributed processing to be performed among SAA-compliant systems, with minimal user intervention. As consistent communications between IBM systems continues to evolve through SAA, this standard ensures the user a framework within which his image system may grow, without concern for inter-system compatibility.
- SNA Type LU 6.2 Command and Printer/Scanner Interfaces provide the user with a consistent means of adding image to existing or new applications and attaching a wide range of scanners and printers to the workstation. These interfaces for IBM SAA ImagePlus Workstation Program/2 Version 1 will be published by IBM.

The IBM ImagePlus Workstation Program Family consists of the DOS-based IBM PS/2 ImagePlus Workstation Program Version 1 and IBM ImagePlus Workstation Program/DOS Versions 2.1 and 2.2, and the OS/2-based IBM SAA ImagePlus Workstation Program/2 Version 1.

Highlights

- High-speed batch scanning with auto indexing may improve productivity by providing capture and indexing capabilities significantly faster than earlier manual approaches.
- National Language Support (NLS) allows international users to support image processing applications in non-English-speaking environments.
- OS/2 capabilities such as Presentation Manager(TM), Communications Manager and multi-tasking, form the basis of a powerful, multi-function, OS/2-based image-processing workstation.

The following functions in the OS/2-based IBM SAA ImagePlus Workstation Program/2 Version 1, can provide practical solutions for new and existing business applications:

- Color and Grayscale support for a variety of paper and photographic applications such as personnel or patient-care records.

- Open Interfaces that allow users and business partners the flexibility to integrate their own applications with image and to attach their own scanners and printers.
- A workstation Import/Export to Disk File feature that gives users the opportunity to exchange information (images, coded data, etc.) among applications such as Office and ImagePlus.

The following features can help users expand their image applications:

- ImagePlus Workstation Program upgradability, migratability and co-existence
- Use of standard IBM image architectures
- System Application Architecture/Common User Access (SAA/CUA) compliance and System Application Architecture/Common Communications Support (SAA/CCS) compliance
- Published interfaces for IBM SAA ImagePlus Workstation Program/2 Version 1

C.30 IBM Workstation AFP View Program

AFP Workbench for Windows provides a significant step towards the integration of applications and data streams on the host and workstation platforms, and extends the usability of the AFP architecture by allowing viewing of the AFPDS data stream on workstations, and by extending the print capability of AFPDS to all workstation printers, not just those supported by and defined to PSF/2.

AFP Workbench for Windows provides the user with the capability of viewing any AFPDS data stream on a workstation using OS/2 or DOS/Windows.

The user can see a good approximation to the final form of the document, limited only by the capabilities of the display screen on the workstation being used.

AFP Workbench for Windows also provides local print capability by using available device drivers. Users can therefore now print on devices not supported by or not defined to PSF/2.

With appropriate device drivers, AFP Workbench for Windows can support local printing on printers using the following data streams:

- AFPDS
- Personal Printer Data Stream
- PostScript
- Printer Control Language

The AFP Workbench for Windows product includes the IBMAFP printer driver to support local printing on AFP devices.

By connecting the printer driver to file instead of a printer, these data streams can be used for further processing as required.

Highlights

- Workstation view of any AFPDS output.
- Local print of AFPDS on any workstation printer.
- Capability of printing only selected pages of an AFPDS document.
- Crop and copy portions of AFPDS pages.
- Capability to convert any AFPDS output to an overlay.
- Support of AFPDS output devices by the provision of the IBMAFP device driver.
- Capability of conversion of AFPDS output to other data streams by connection of available device drivers to file.

C.31 Ami Pro

Ami Pro is a product of the Lotus Development Corporation, and is used extensively on Personal Computers as a high function word processor.

Ami Pro is also notable for the number of products it can import word processing file from, and export them to.

Ami Pro's import and export capabilities include:

Table 15 (Page 1 of 2). Ami Pro Import/Export

File Type	Import	Export
Advance Write	Y	Y
ASCII	Y	Y
dBase	Y	N
DCA/FFT	Y	Y
DCA/RFT	Y	Y
DIF**	Y	N
DisplayWrite 4	Y	Y
Enable	Y	N
Excel	Y	N
Exec MemoMaker	Y	Y
Manuscript	Y	Y
Microsoft Word	Y	Y
Multimate	Y	Y
Navy DIF	Y	Y
Office Writer	Y	Y
Paradox**	Y	N
Peach Text	Y	Y
Professional Write**	Y	Y
Rich Text Format (RTF)	Y	Y
Samna Word**	Y	Y
SmartWare	Y	N

Table 15 (Page 2 of 2). Ami Pro Import/Export		
File Type	**Import**	**Export**
Symphony**	Y	N
Windows Write	Y	Y
Word for Windows	Y	Y
WordPerfect	Y	Y
WordStar**	Y	Y
WordStar 2000	Y	Y
Note: **Y** Yes **N** No Note that not all characteristics will be preserved in all import or export movements. Where it is critical that a particular function is preserved, **check it first**.		

C.32 Corel Draw

Corel Draw is a product of the Corel Systems Corporation, and is used extensively on Personal Computers to create high quality graphics images.

The Corel Draw package includes a program called *Corel Trace*, which allows bitmaps to be converted to vector graphics. This can be a very useful facility, to allow easy modification of graphics, to save storage, and to allow transforms that could not be otherwise performed.

Corel Draw is also notable for the number of file types it can import graphics and images from, and export graphics and images to.

Corel Draw's import and export capabilities include:

Table 16. Corel Draw Import/Export		
File Type	**Import**	**Export**
Computer Graphics Metafile(CGM)	Y	Y
Autocad DXF Interchange file(DXF**)	P	P
Encapsulated PostScript(EPS)	P	Y
Digital Research GEM format(GEM**)	Y	Y
"Graphics Language" (plot file)(GL)	Y	Y
Apple PICT file(PCT)	Y	Y
Paintbrush PCX file(PCX)	Y	M
Picture Interchange File(PIF)	Y	Y
Tagged Image Format(TIF)	Y	M
Note: **Y** Yes **N** No **M** Monochrome only **P** Only partial conversion		

C.33 Freelance

Freelance is a product of the Lotus Development Corporation, and is used extensively on Personal Computers to create good quality business graphics and charts.

Freelance can export the widely used CGM standard of graphics file, which allows it to share its output across the workstation and mainframe world. When moving files into the mainframe world, a special profile for conversion to GDF exists in GDDM, *CGMFP2* for Freelance version 2, and *CGMFP3* for Freelance version 3. Use of these profiles will ensure the most faithful conversion of Freelance colors into the mainframe environment.

Otherwise, Freelance CGM files can be treated in exactly the same way as Corel Draw CGM files (see 13.1, "Getting Pictures into IBM BookManager from Corel Draw" on page 103). Freelance CGM files can be imported into Corel Draw if they have to be converted into a format that Freelance doesn't support.

C.34 OS/2 Image Support

The OS/2 Image Support program enables users to create color, greyscale, and bilevel images using a scanner or video adapter.

Functions provided for working with images:

- Modify
- Print
- Change
- Format
- Display
- Convert

From the OS/2 clipboard images can be:

- Stored
- Retrieved

OS/2 Image Support:

- Uses the Presentation Manager environment of OS/2 Standard Edition Version 1.2 or OS/2 Extended Edition Version 1.2
- Scanner support via IBM 3119 PageScanner, IBM 3118 Scanner and complementary scanners.
- Prints on all-points-addressable printers supported by OS/2 PM and on complementary printers.
- File formats supported are:
  - IOCA
  - TIFF
  - MO:DCA-P
  - RFT:DCA
  - PIC
  - GIF
  - CGM
  - and others
- OS/2 Image Support is accessed from:
  - Main OS/2 OfficeVision
  - Window via an image icon

  - Being invoked automatically
- Via OfficeVision/2 communication links, images may be stored and retrieved from:
  - OS/400
  - MVS host
  - VM host
  - Local area network (LAN) server
- Creates images that can be used by other programs, for example,
  - OfficeVision/2
  - Storyboard Plus
  - OS/2 Office editor
  - ImagePlus Workstation/2
  - Facsimile Support/400*
  - GDQF
  - IBM CAD
  - Corel Draw
  - Microsoft Windows 3.0 Paintbrush**,
  - Interleaf Publishing
  - Image-capable DisplayWrite products

C.35 MARKUP

The MARKUP program contains an entry-assist program that helps you enter GML document elements. Tags and editing commands can be selected from a menu or specially-designated keys can be used to enter the tags and editing commands. In either case, MARKUP provides interactive feedback to help you structure your document and enter GML elements correctly.

The MARKUP program also contains a separate customization program called TAILOR. This program determines how GML tags work with the text on the MARKUP screen and allows you to customize keyboard and screen features.

The tags customized with TAILOR must also be recognized by DCF to format and print on the host.

MARKUP'S two programs combine to give you the flexibility to create different types of documents supported by your host processor.
For example, you can create and customize:

- Reports
- Memos
- Manuals
- Books
- Parts Listings
- Directories

MARKUP provides context-sensitive, online help for information about program functions. MARKUP also supports locally attached IBM printers allowing you to print draft copies of documents as they appear on the MARKUP screen.

MARKUP is designed as a productivity aid for business professionals who are conversant with GML but who are not proficient GML users.

C.36 IBM PS/2 Image Adapter/A

Available Versions:

- The IBM PS/2 Image Adapter/A 1MB
- The IBM PS/2 Image Adapter/A 3MB
- The IBM PS/2 Image Adapter/A 3MB 6091

Highlights

- Supports high speed compression/decompression algorithms that contribute to rapid response times for printing, scanning, or displaying image information.
- Supports maximum resolutions of 1600 x 1200 monochrome and 1280 x 1024 color.
- Provides upgrade options to enable business growth and protect the customers investment in base components.
- Supports up to 256 colors from a palette of 16 million and up to 256 grayscales, for a broad range of image and technical professional applications.
- Provides compatibility with VGA and 8514/A modes.
- Supports all IBM PS/2 displays plus 8506/8 and 6091-019.
- Has high performance processor and built-in hardware assist features.
- Features 16 Bit data transfer path to PS/2 Micro Channel bus.

C.37 DisplayWrite/370

C.37.1 Description

DisplayWrite/370 is a System /370 host text processing application package which includes functions for document creation and revision, interactive document formatting, printing, and linguistic support.

Highlights

- Full screen interactive text editor/formatter
- Complete context dependent help and tutorial
- Command lists to perform special tasks
- Basic text entry and editing capabilities
- Advanced text editing capabilities
- Support of RFT:DCA and FFT:DCA document interchange
- National Language Support
- Multi-language linguistic aids

C.37.2 Compatibility with Document Composition Facility (DCF)

DisplayWrite/370 provides an interactive formatting function, whereby the "finished" view of the document is directly presented to the user. Formatting functions which require two-pass processing, such as table of contents, indexing or footnote references will not be supported. These functions may be handled by Document Composition Facility.

Under VM/SP, DisplayWrite/370 enables the user to convert most of the RFT:DCA controls into SCRIPT controls. This conversion route has to be handled with care. SCRIPT controls are format specific, and the preferred conversion is to device and application independent GML.

C.37.3 Document Interchange

DisplayWrite/370 documents can be interchanged in either revisable form (RFT:DCA) or final form (FFT:DCA) by all products supporting the Document Content Architecture.

- DisplayWrite/370 makes no distinction between the document origination and document types. Whether it is created by DisplayWrite/370 or it is received from other systems or products using the Document Content Architecture standard set of controls, the same mode of operation is provided.
- Any RFT:DCA defined structure fields, structures, single- or multi-byte controls, and/or the parameter setting in these controls, if not supported by DisplayWrite/370, will be kept intact in the data stream, so as to preserve the data stream integrity as required by the Document Content Architecture.

C.37.4 Printer Support

Printer support is depending on the facilities provided by the program that invokes DisplayWrite/370. Under MVS/SP or VSE/SP, DisplayWrite/370 can create the following data stream:

- SNA Character String (SCS)
- IBM 5210 (using SCS in CICS environment)

Under VM/SP, DisplayWrite/370 can create the following data streams:

- FFT:DCA
- 1403DS
- 3800 Model 1

In the VM environment, there is no direct printer support. It is expected that the file created via the transform process will be printed on an appropriate printer as supported by the operating system spool service.

By transforming the DisplayWrite/370 document to DCF, it can be printed on a high-quality printer.

C.38 OfficeVision/MVS

IBM OfficeVision/MVS (OV/MVS) is the MVS member of the IBM OfficeVision Family of office systems applications. OfficeVision/MVS provides a set of office functions for non-programmable terminals and IBM Disk-Operating (DOS) programmable workstations. OfficeVision/MVS Release 2.0 enhances Release 1.0 by adding an Enterprise Address Book, an Entry Level Enrollment Option, toleration of Double Byte Character Set (DBCS) information, and usability, performance, and installation enhancements.

Document writing feature enhancements Provides the ability, via the Application Connectivity Feature (ACF), to transfer TSO Graphic Data files (GDF) and store these files in the ADMF data set of GDDM, or to incorporate them through DW/370 into an RFT document. This capability requires

the Document Writing Feature and DW/370 (DW/370 Version 2 or DW/370 Image and Graphics Feature).

C.39 IBM SAA OfficeVision/400 Version 2

IBM SAA OfficeVision/400 Version 2 provides a rich set of office functions for document preparation, filing and retrieval of information, communication of information and time management.

Documents can be prepared using the OfficeVision/400 Editor or one of the DisplayWrite Family of editors if using the PC Support/400 licensed program. In addition, PC Support/400 Version 2 Release 1 Modification 1 provides a tool to allow the use of a non-IBM editor as "editor of choice". This tool is contained in a PC Support Tools folder that is shipped with PC Support/400 and does not receive full support. Refer to Programming Announcement ZP91-0245 dated April 22, 1991 for more information regarding the PC Support Tools folder.

Proofreading aids are provided and use the optional IBM Language Dictionaries/400 licensed program and/or user-created permanent dictionaries.

Information can be filed into a document library. The library can contain documents, PC files, images and graphics. A user can retrieve information from the library by specifying a number of search parameters or document descriptors. Using Version 2 Release 1 Modification 1, users will be able to specify a text search that will result in full-text retrieval of RFT:DCA and FFT:DCA documents which have been indexed for this type of search.

Information can be exchanged between OfficeVision/400 users and with users in other environments in the OfficeVision Family. This information includes documents, PC files and notes. Notes and documents may also be exchanged through TCP/IP and through X.400. Exchange with X.400 is available in V2 R1.1. Exchange with OV/VM, TCP/IP and X.400 will require additional software.

C.40 IBM Presentation Manager Office/2

PMO/2 functions are detailed as follows:

Access to incoming electronic mail. An overview of the in-basket is transferred to OS/2 and presented through a window, where each element can be selected for further processing through a separate window. Pull-down menus control further processing of the elements, for example, reply to incoming mail, printing and deletion. The user has the option to make notes, documents and files available for further processing on the host or workstation.

Out-going electronic mail. Through a simple icon interface notes, documents, and files can be sent from the OS/2 workstation. All types of documents and files from the workstation can be mailed via the host system's electronic mail functions. Notes can be created and mailed individually or in conjunction with a document or file.

Address book on OS/2. The option of using a set of address books to keep track of personal correspondence can be used when creating outgoing mail.

Access to files on the host system. Windows are created showing documents and files on the host system via an easy-to-use icon interface. As with the in-basket, the individual elements can be selected for further processing in a separate window. Using another icon OS/2 files and documents are transferred to the host file cabinet.

Access to host system functions. Other functions on the host system are accessed via 3270 emulation which can also be activated via the icon interface.

C.41 Application Area Summary

	Office Vision	ImagePlus	Production Publishing
generates: output data stream normal tasks:	Text Image Graphics RFT:DCA Edit Display Print Distribute	Text Image MO:DCA-P Display Print Retrieve	Text Image Graphics GML SGML Edit Format Display Print Distribute
Software for:	Edit DW/370 Display BrowseMaster Print PSF GDDM Distribute SNA File transfer	Retrieve ImagePlus Display ImagePlus Print PSF PSF/2	Edit Text Editor Format DCF Display BrowseMaster Print PSF PSF/2 GDDM Distribute SNA File transfer
Software overview:	OfficeVision/2 LAN OfficeVison/MVS OfficeVison/VM OfficeVison/400	MVS/ESA Folder Appl. Facility Object Distr. Mgr. AS/400 Workfolder Appl. Facility/400 Workstation Workstation Prog./DOS Workstation Prog./2	Production publishing BookMaster BrowseMaster ProcessMaster ProcessMaster CALS SGML Translator TextTagger BookManager Professional publising IBM Interleaf Publisher - PS/2 Interleaf RS/6000 Personal publishing PostScript printers Compl. Software PostScript Interpreter for AFP

Figure 36. Application Area Summary. A summary of the major functions and software by application area.

Appendix D. Products Involved in Printing and Viewing

This appendix contains tables showing IBM products involved in Printing and Viewing. They have essentially been taken directly from *SAA Common Communications Support Summary, GC31-6801.*

D.1 Product and Operating System Tables

The following tables list the products and platforms that implement the Common Communications Support architectures of SAA. The tables do not specify the portions of an architecture that a product may implement or indicate how data may be interchanged among implementing products. For information about interchange requirements, refer to the appropriate topic in this document.

Each table has the headings **MVS, VM, OS/400, OS/2, IMS,** and **CICS**. When products in one of these operating environments implement an architecture, the product names are listed. When the operating environment itself implements an architecture, the operating environment name is shown.

D.1.1 BCOCA

MVS	VM	OS/400	OS/2	IMS	CICS
PSF 3812-2[7] 3816[7] 4028[7] 4224[7] 4230[7] 4234[7]	PSF 3812-2[7] 3816[7] 4028[7] 4224[7] 4230[7] 4234[7]	AFPU[6] FormsX[6] OS/400 3812-2[7] 3816[7] 4028[7] 4224[7] 4230[7] 4234[7]	PSF 3812-2[7] 3816[7] 4028[7]	PSF 3812-2[7] 3816[7] 4028[7] 4224[7] 4230[7] 4234[7]	AIForm[6] PSF 3812-2[7] 3816[7] 4028[7] 4224[7] 4230[7] 4234[7]

D.1.2 CDRA

MVS	VM	OS/400	OS/2	IMS	CICS
DB2	SQL/DS	OS/400 CIM S/400	DDCS/2	DB2	DB2

D.1.3 DIA

MVS	VM	OS/400	OS/2	IMS	CICS
PS/TSO		AS/400 Ofc OS/400 OV/400	OV/2		DISOSS OV/MVS

[6] Generator — creates data streams.

[7] Receiver — interprets and processes data streams.

[8] Refers to the Record Level Input/Output (RLIO) and Distributed FileManager (DFM) facilities of the File Resource Manager component of Data Facility Distributed Storage Manager. DFM is source only.

D.1.4 FD:OCA

MVS	VM	OS/400	OS/2	IMS	CICS
DB2	SQL/DS	CIM S/400 OS/400	DDCS/2	DB2	DB2

D.1.5 FOCA

IPDS, MO:DCA, and RFT:DCA use FOCA defined parameters. Therefore, all products that implement these architectures also implement FOCA.

D.1.6 GOCA

MVS	VM	OS/400	OS/2	IMS	CICS
DW/370[9] GDDM OSP PSF 3812-27 38167 38257, [10] 38277, [10] 38287 38317, [10] 38357, [10] 40287 42247 42307 42347	DW/370[9] GDDM OSP PSF 3812-27 38167 38257, [10] 38277, [10] 38287 38317, [10] 38357, [10] 40287 42247 42307 42347	AS/400 Ofc OS/400 OV/400 3812-27 38167 38257, [10] 38277, [10] 38317, [10] 38357, [10] 40287 42247 42307 42347	DW5/2 DW5/2C OS/2 EE OV/2 PSF 3812-27 38167 38257, [10] 38277, [10] 38287 38317, [10] 38357, [10] 40287	GDDM PSF 3812-27 38167 38257, [10] 38277, [10] 38287 38317, [10] 38357, [10] 40287 42247 42307 42347	DW/370[9] GDDM OSP PSF 3812-27 38167 38257, [10] 38277, [10] 38287 38317, [10] 38357, [10] 40287 42247 42307 42347

D.1.7 IOCA

D.1.7.1 FS10

MVS	VM	OS/400	OS/2	IMS	CICS
DW/370[9] GDDM IHF/V2 ImagePlus IVU OSP PSF 3812-27 38167 38257, [10] 38277, [10] 38287 38317, [10] 38357, [10] 40287	DW/370[9] GDDM IHF/V2 IVU OSP OV/VM PROFS PSF 3812-27 38167 38257, [10] 38277, [10] 38287 38317, [10] 38357, [10] 40287	AS/400 Ofc ImagePlus OS/400 OV/400 3812-27 38167 38257, [10] 38277, [10] 38317, [10] 38357, [10] 40287	DW5/2 DW5/2C ImagePlus OS/2 EE OS/2 IS PSF 3812-27 38167 38257, [10] 38277, [10] 38287 38317, [10] 38357, [10] 40287	GDDM ImagePlus PSF 3812-27 38167 38257, [10] 38277, [10] 38287 38317, [10] 38357, [10] 40287	DISOSS DW/370[9] GDDM ImagePlus IVU OSP PSF 3812-27 38167 38257, [10] 38277, [10] 38287 38317, [10] 38357, [10] 40287

[9] Image and Graphics Feature.

[10] With the Advanced Function Image and Graphics (AFIG) feature.

D.1.7.2 FS11

MVS	VM	OS/400	OS/2	IMS	CICS
ImagePlus		ImagePlus	ImagePlus	ImagePlus	ImagePlus

D.1.8 IPDS

MVS	VM	OS/400	OS/2	IMS	CICS
GDDM[6]	GDDM[6]	OS/400[6]	PSF[6, 7]	GDDM[6]	GDDM[6]
PSF[6]	PSF[6]	RPM	3812-2[7]	PSF[6]	PSF[6]
RPM[6, 7]	RPM[6, 7]	3812-2[7]	3816[7]	RPM[6, 7]	RPM[6, 7]
3812-2[7]	3812-2[7]	3816[7]	3820[7]	3812-2[7]	3812-2[7]
3816[7]	3816[7]	3820[7]	3825[7]	3816[7]	3816[7]
3820[7]	3820[7]	3825[7]	3827[7]	3820[7]	3820[7]
3825[7]	3825[7]	3827[7]	3828[7]	3825[7]	3825[7]
3827[7]	3827[7]	3831[7]	3831[7]	3827[7]	3827[7]
3828[7]	3828[7]	3835[7]	3835[7]	3828[7]	3828[7]
3831[7]	3831[7]	4028[7]	3900[7]	3831[7]	3831[7]
3835[7]	3835[7]	4224[7]	4028[7]	3835[7]	3835[7]
3900[7]	3900[7]	4230[7]		3900[7]	3900[7]
4028[7]	4028[7]	4234[7]		4028[7]	4028[7]
4224[7]	4224[7]			4224[7]	4224[7]
4230[7]	4230[7]			4230[7]	4230[7]
4234[7]	4234[7]			4234[7]	4234[7]

D.1.9 MO:DCA

D.1.9.1 MO:DCA-P IS/1

MVS	VM	OS/400	OS/2	IMS	CICS
ImagePlus[6] PSF[7]	PSF[7]	ImagePlus[6] OS/400[7]	ImagePlus[6] PSF[7]	ImagePlus[6] PSF[7]	ImagePlus[6] PSF[7]

D.1.9.2 MO:DCA-P IS/2

MVS	VM	OS/400	OS/2	IMS	CICS
ImagePlus[6, 7]		ImagePlus[6, 7]	ImagePlus[6, 7]	ImagePlus[6, 7]	ImagePlus[6, 7]

D.1.9.3 MO:DCA-L

MVS	VM	OS/400	OS/2	IMS	CICS
			AConnS[7] OS/2 EE[6, 7] OS/2 IS[7] PSF[7]		

D.1.10 PTOCA

MVS	VM	OS/400	OS/2	IMS	CICS
DW/370 ImagePlus PSF 3812-2[7] 3816[7] 3820[7] 3825[7] 3827[7] 3828[7] 3831[7] 3835[7] 3900[7] 4028[7] 4224[7] 4230[7] 4234[7]	DW/370 PSF 3812-2[7] 3816[7] 3820[7] 3825[7] 3827[7] 3828[7] 3831[7] 3835[7] 3900[7] 4028[7] 4224[7] 4230[7] 4234[7]	ImagePlus OS/400 3812-2[7] 3816[7] 3820[7] 3825[7] 3827[7] 3831[7] 3835[7] 4028[7] 4224[7] 4230[7] 4234[7]	ImagePlus PSF 3812-2[7] 3816[7] 3820[7] 3825[7] 3827[7] 3828[7] 3831[7] 3835[7] 3900[7] 4028[7]	ImagePlus PSF 3812-2[7] 3816[7] 3820[7] 3825[7] 3827[7] 3828[7] 3831[7] 3835[7] 3900[7] 4028[7] 4224[7] 4230[7] 4234[7]	DW/370 ImagePlus PSF 3812-2[7] 3816[7] 3820[7] 3825[7] 3827[7] 3828[7] 3831[7] 3835[7] 3900[7] 4028[7] 4224[7] 4230[7] 4234[7]

D.1.11 RFT:DCA

MVS	VM	OS/400	OS/2	IMS	CICS
DW/370 PMaster	DW/370 PMaster	AS/400 Ofc OV/400	DW5/2 DW5/2C OV/2		DISOSS DW/370

D.1.12 3270DS

MVS	VM	OS/400	OS/2	IMS	CICS
GDDM TSO/E	CICS GDDM	OS/400[11]	OS/2 EE	GDDM IMS	CICS GDDM

[11] AS/400 supports the 3270 Data Stream from 3174 controllers attached via a remote line or token ring. Conversion to and from the 5250 data stream is handled internally in the OS/400. The OS/400 can pass a 3270 data stream through to a System/370 computer.

Appendix E. Bibliography

E.1 IBM Publications

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

Working with this book we recommend a close look the the Red books mentioned at the end of this chapter.

E.1.1 Architecture

- Data Stream and Object Architectures: Mixed Object Document Content Architecture Reference, SC31-6802
- Data Stream and Object Architectures: Presentation Text Object Content Architecture Reference, SC31-6803
- Data Stream and Object Architectures: Graphics Object Content Architecture Reference, SC31-6804
- Data Stream and Object Architectures: Image Object Content Architecture Reference, SC31-6805
- Advanced Function Printing Data Stream Reference, S544-3202
- Font Object Content Architecture Reference, S544-3285
- Bar Code Object Content Architecture Reference, S544-3766
- Character Data Representation Architecture—Level 1 Reference, SC09-1390
- Formatted Data Object Content Architecture Reference, SC31-6806
- Document Content Architecture: Revisable-Form Text Reference, GC23-0758
- Information Interchange Architecture Concepts, GG24-3503

E.1.2 Publishing Systems ProcessMaster VM Edition

- IBM Publishing Systems ProcessMaster VM Edition General Information, GC34-5031
- IBM Publishing Systems ProcessMaster VM Edition Administrator's Guide, SC34-5034
- IBM Publishing Systems ProcessMaster VM Edition User's Guide, SC34-5033
- IBM Publishing Systems ProcessMaster VM Edition Editing, SC34-5035
- IBM Publishing Systems ProcessMaster VM Edition Character Graphics Editing, SC34-5036
- IBM Publishing Systems ProcessMaster VM Edition Reference Summary, SC34-5037
- IBM Publishing Systems ProcessMaster VM Edition Critique Guide, SC34-5139
- IBM Publishing Systems ProcessMaster Workstation Edition User's Reference, SC34-5050

E.1.3 Publishing Systems ProcessMaster MVS Edition

- IBM Publishing Systems ProcessMaster MVS Edition Licensed Program Specifications, GC34-5090
- IBM Publishing Systems ProcessMaster MVS Edition General Information, GC34-5091
- IBM Publishing Systems ProcessMaster MVS Edition User's Guide, SC34-5092

- IBM Publishing Systems ProcessMaster MVS Edition Administrator's Guide, SC34-5093
- IBM Publishing Systems ProcessMaster MVS Edition Reference Summary, SC34-5094
- IBM Publishing Systems ProcessMaster Workstation Edition User's Reference, SC34-5050

E.1.4 Publishing Systems TextTagger

- IBM Publishing Systems ProcessMaster TextTagger Feature User's Guide and Reference, SC34-5142
  The following publication is shipped with the product and is not available separately.
- IBM Publishing Systems TextTagger Workstation Edition User's Guide and Reference

E.1.5 Document Composition Facility

- Document Composition Facility and Document Library Facility General Information Manual, GH20-9158
- Document Composition Facility: SCRIPT/VS Language Reference, SH35-0070
- Document Composition Facility: Starter Set User's Guide, SH20-9186
- Document Composition Facility: GML Starter Set Reference, SH20-9187
- Document Composition Facility: Messages, SH35-0048
- Document Composition Facility: SCRIPT/VS Text Programmer's Guide, SH35-0069
- DCF Post Processor Examples, S544-3484

E.1.6 Document Composition Facility - Office Document Feature

- Document Composition Facility: Office Document Feature User's Guide, G544-3129
- Document Composition Facility: Office Document Feature Reference, S544-3130

E.1.7 Publishing Systems BookMaster

- IBM Publishing Systems BookMaster General Information, GC34-5006
- IBM Publishing Systems BookMaster User's Guide, SC34-5009
- IBM Publishing Systems BookMaster Creating Named Styles, SC34-5008

E.1.8 Publishing Systems BrowseMaster

- IBM Publishing Systems BrowseMaster V2 Installing and Using, SH23-0016

E.1.9 Publishing Systems DrawMaster

- IBM Publishing Systems DrawMaster General Information, GC34-5021
- IBM Publishing Systems DrawMaster User's Guide and Reference, SC34-5022
- IBM Publishing Systems DrawMaster Introduction, SC34-5023
- IBM Publishing Systems DrawMaster Quick Reference, SC34-5024

E.1.10 Publishing Systems PostScript Interpreter

- IBM Publishing Systems PostScript Interpreter for Advanced Function Printing MVS User's Guide, SC34-5112
- IBM Publishing Systems PostScript Interpreter for Advanced Function Printing VM User's Guide, SC34-5082

E.1.11 BookManager - VM

- BookManager READ/VM and BookManager BUILD/VM General Information, GC23-0447
- BookManager READ/VM: Getting Started and Command Summary, SC23-0448
- BookManager READ/VM: Displaying Online Books, SC23-0449
- BookManager READ/VM: Installation and Customization, SC23-0455
- BookManager BUILD/VM: Preparing Online Books, SC23-0450
- BookManager BUILD/VM: Installation and Customization, SC23-0451
- BookManager READ/VM and BookManager BUILD/VM: Directory of Programming Interfaces, GC23-0453

E.1.12 BookManager - MVS

- BookManager READ/MVS and BookManager BUILD/MVS General Information, GC38-2032
- BookManager READ/MVS: Getting Started and Command Summary, SC38-2033
- BookManager READ/MVS: Displaying Online Books, SC38-2034
- BookManager READ/MVS: Installation and Customization, SC38-2035
- BookManager BUILD/MVS: Preparing Online Books, SC38-2036
- BookManager BUILD/MVS: Installation and Customization, SC38-2037

E.1.13 SGML Translator DCF Edition

- IBM Publishing Systems SGML Translator DCF Edition General Information, GC34-5071
- IBM Publishing Systems SGML Translator DCF Edition Programming Guide and Reference, SC34-5072
- IBM Publishing Systems SGML Translator DCF Edition Validating and Translating an SGML Document, SC34-5074
- IBM Publishing Systems SGML Translator DCF Edition Creating a DTD, SC34-5075
- IBM Publishing Systems SGML Translator DCF Edition Creating an SGML Document for DCF Processing, SC34-5076

E.1.14 CALS

- IBM Publishing Systems ProcessMaster CALS Application Feature VM User's Guide, SC34-5150
- IBM Publishing Systems ProcessMaster CALS Application Feature Creating CALS Documents, SC34-5151
- IBM Publishing Systems CALS Glossary, SC34-5152
- IBM Publishing Systems ProcessMaster CALS Application Feature MVS User's Guide, SC34-5154

E.1.15 SGML TextWrite

The following publications are shipped with the product and are not available separately.

- SGML TextWrite OS/2 Edition User's Guide and Reference
- SGML TextWrite OS/2 Edition CALS Application User's Guide and Reference
- SGML TextWrite OS/2 Edition General Document Application User's Guide and Reference
- SGML TextWrite Tools OS/2 Edition User's Guide and Reference
- SGML TextWrite Tools OS/2 Edition Application Worksheet Packet

E.1.16 Image Handling Facility

- Introducing Image Handling Facility Version 2 Messages, GH12-5278
- Image Handling Facility Version 2 Messages, SH12-5282
- Getting Started with Image Handling Facility Version 2 Tutorial, SH12-5279
- Using Image Handling Facility Version 2, SH12-5280
- Image Handling Facility Version 2 Installation and Administration, SH12-5281

E.1.17 GDDM

- GDDM General Information GC33-0319
- GDDM Release Guide GC33-0320
- GDDM Library Guide and Master Index GC33-0595
- GDDM Image View Utility SC33-0479
- GDDM-REXX Guide SC33-0478
- GDDM Interactive Map Definition SC33-0338
- GDDM Guide for Users SC33-0327
- GDDM-PGF Interactive Chart Utility SC33-0328
- GDDM Image Symbol Editor SC33-0329
- GDDM-PGF Vector Symbol Editor SC33-0330
- GDDM Typefaces and Shading Patterns, SC33-0554
- GDDM-PCLK Reference Summary SX33-6067
- GDDM-CSPF User's Guide SC33-0552
- GDDM Application Programming Guide SC33-0337
- GDDM Base Programming Reference SC33-0332
- GDDM Base Programming Reference Summary SX33-6053
- GDDM-PGF Programming Reference SC33-0333
- GDDM-PGF Programming Reference Summary SX33-6054
- GDDM-GKS Programming Guide and Reference SC33-0334
- GDDM Installation and System Management for MVS, GC33-0321
- GDDM Installation and System Management for VSE, GC33-0322
- GDDM Installation and System Management for VM, GC33-0323
- GDDM Performance Guide, SC33-0324
- GDDM Messages, SC33-0325
- GDDM Diagnosis and Problem Determination Guide, SC33-0326

E.1.18 GDQF

- Graphical Display and Query Facility General Information Manual, GH20-6223
- Graphical Display and Query Facility Using GDQF Base Version 2 Release 1.0 (VM), SH52-0252
- Graphical Display and Query Facility Using GDQF Base Version 2 Release 1.0 (MVS), SH52-0253

E.1.19 Image

- IBM ImagePlus Folder Application Facility MVS/ESA Application Program Interface Version 2 Application Programmers Guide, SC31-7531
- MVS/ESA Folder Application Facility, Programmers Guide Release 2, SC38-2013
- AS/400 Workfolder Application Facility Application Programming Interfaces, GC38-3034
- IBM SAA ImagePlus Workfolder Application Facility/400 Programming Interfaces Version 2 Release 1, SC38-3049
- IBM SAA ImagePlus Workstation Program/2 Programmers Guide, SC09-1300

E.1.20 Office

- AS/400 Office: Application Programming Interface Integration Guide for Programmers, GG22-9442
- AS/400 Office Application Programming Interface Integration Guide, GG24-9442
- OS/2 Office Application Integration, SH21-0447
- Programming Interfaces for OfficeVision/MVS, SH21-0531
- OfficeVision/VM Programmers Guide, SH21-0581
- Using CLISTS with DisplayWrite/370, SH12-5196

E.1.21 OS/2 Image Support

- Introducing OS/2 Image Support, GX09-1213
- Installing OS/2 Image Support, SC09-1334
- OS/2 Image Support V1.1, G221-2661

E.1.22 ISO Standards

- International Standard ISO 8879: Information processing--Text and office systems--Standard Generalized Markup Language (SGML)
- International Draft Standard ISO DIS 10179: Document Style Semantics and Specification Language (DSSSL)
- International Draft Standard ISO DIS 10180: Standard Page Description Language (SPDL)
- International Standard ISO 8613: Office Document Architecture (ODA)

E.1.23 ITSO Red Books

- ITSC IBM Architectures, GG24-3503
- ITSC Document Transforms Cookbook, GG24-3530
- ITSC PSF/VSE V2.1, GG24-3638
- ITSC RPM V3 Implementation Guide, GG24-3620
- ITSC Document Processing Guidelines, GG24-3659
- ITSC Distributed Print, GG24-3766
- ITSC Office Systems Primer, GG24-1635
- ITSC Printing in a VM Office Environment: End User's Guide, GG24-3830
- ITSC DW/370 V2 in OV/MVS R2, GG24-3683
- ITSC OV/400 Printing, GG24-3697
- ITSC AS/400 Printing II, GG24-3704
- ITSC FAX Support/400, GG24-3797
- ITSC OS/2 V2.0 Vol 5: Print Subsystem (ITSC), GG24-3775
- ITSC Printing PostScript Language, GG24-3529

E.1.24 Other Documentation

- Applications System/400 Advanced Function Printing by Example, SC21-8181
- SAA Common System Programming Inferface: Presentation Reference, SC26-4359
- IBM Distributed Printing Solution, G511-1725
- IBM Print Services Facility/2, G511-1726
- Page Printer Migration Programming Guide, S544-3228
- Host Font Data Stream Reference, S544-3289
- Intelligent Printer Data Stream Reference, S544-3417

E.2 Workstation Documents

- Microsoft Windows User's Guide, Z85F-1687-00
- Corel Draw User's Guide
- Ami Pro User's Guide
- HP PCL 5 Printer Language Technical Reference Manual, 33459-90903
- Aldus TIFF Developer's Toolkit, Aldus Corporation

Glossary

This glossary includes definitions from the following sources:

- Definitions reprinted from the *American National Dictionary for Information Processing Systems* are identified by the symbol (A) following the definition.
- Definitions reprinted from a published section of the International Organization for Standardization's *Vocabulary—Information Processing* or from a published section of the ISO *Vocabulary—Office Machines* are identified by the symbol (I) following the definition. Because many ISO definitions are also reproduced in the *American National Dictionary for Information Processing Systems*, ISO definitions may also be identified by the symbol (A).
- Definitions reprinted from working documents, draft proposals, or draft international standards of ISO Technical Committee 97, Subcommittee 1 (Vocabulary) are identified by the symbol (T) following the definition, indicating that final agreement has not yet been reached among its participating members.
- Definitions that are specific to IBM products are so labeled, for example, "In SNA," or "In VM."

References

The following cross-references are used in this glossary:

Contrast with. This refers to a term that has an opposite or substantively different meaning.

Synonym for. This indicates that the term has the same meaning as a preferred term, that is also defined
in this glossary.

Synonymous with. This is a backward reference from a defined term to all other terms that have the same meaning.

See. This refers to multiple-word terms that have the same last word.

See also. This refers to related terms that have a related, but not synonymous, meaning.

Deprecated term for. This indicates that the term should not be used. It refers to a preferred term, that is also defined in this glossary.

A

ABIC. Adaptive Bilevel Image Compression

Advanced Function Printing. A set of licensed programs that use the all-points-addressable concept to print text and graphics on a printer.

Advanced function printing data stream. The printer data stream used for printing advanced function printing data. The AFPDS includes composed text, page segments, electronic overlays, form definitions, and fonts that are downloaded from the system.

Advanced printer function. A function of the AS/400 Application Development Tools licensed program that allows a user to design symbols, logos, special characters, large characters, and forms tailored to a business or data processing application, and supports printing of any design on the 5224 or 5225 dot matrix printer.

Advanced program-to-program communications. Data communications support that allows programs on an AS/400 system to communicate with programs on other systems having compatible communications support. APPC is the AS/400 method of using the SNA LU session type 6.2 protocol.

AFIG. Advanced Function Image and Graphics Feature

AFP. Advanced Function Printing

AFPDS. Advanced Function Printer Data Stream

AFPU. Advanced Function Printing Utilities/400

All points addressable. In computer graphics, pertaining to the ability to address and display or not display each picture element (pel) on a display surface.

Alphanumeric. Pertaining to a character set that contains letters, digits, and other characters such as punctuation marks. (A). Synonymous with alphanumeric.

American National Standards Institute. An organization for the purpose of establishing voluntary industry standards.

ANSI. American National Standards Institute see also American National Standards Institute

APA. All points addressable

APA printers. Devices that are all points addressable; in other words, devices that print with picture elements on the printing medium at any valid location on a sheet of paper.

APF. advanced printer function

APPC. Advanced Program-to-Program Communications

AS/400 Ofc. AS/400 Office

AS/400. Application System/400

ASCII. American Standard Code for Information Interchange; standardized format and collating sequence for internal representation of characters, used by a number of operating systems including IBM Operating System/2.

B

Bar code. A code representing characters by sets of parallel bars of varying thickness and separation that are read optically by transverse scanning. (I)

BCD1. A subset of the full capabilities of BCOCA

BCOCA. Bar Code Object Content Architecture

BGU. Business Graphics Utilities

Boldface. (1) A heavy-faced type. (2) Printed in heavy-faced type

Bounded-box format. An organization of character graphics and information used by advanced-function-printing programs for printing on printers such as the IBM 3820 and 3827. Character boxes containing each character graphic do not require character-positioning information in the form of untoned pels in the character boxes. In addition, a single character set can be used for all combinations of character rotation and text orientation. Fonts in bounded-box format usually require less storage than fonts in unbounded-box format. Contrast with unbounded-box format.

C

CAD. Computer-aided Design

CADAM. Computer-graphics Augmented Design and Manufacturing System. CADAM is an interactive graphics system used for two- or three-dimensional design and drafting.

CAD/CAM. Computer-aided design/computer-aided manufacturing.

CAEDS. Computer Aided Engineering Design Systems. CAEDS is an integrated design system for solving mechanical design and analysis problems.

CALS. Computer-aided Acquisition and Logistic Support

CATIA. Computer-graphics Aided Three-dimensional Interactive Application

Camera-ready master. Text and graphics merged on a page, ready for printing.

CCA. Character Content Architecture

CCITT. International Telegraph and Telephone Consultative Committee

CCS. Common Communication Support

CCSID. Coded Character Set Identifier

CDPF. Composed Document Printing Facility. An IBM program product that allows the user to produce high-quality, high-resolution, "camera ready" master pages.

CDPDS. Composed Document Printer Data Stream (GDDM)

CDRA. Character Data Representation Architecture

CGM. Computer Graphics Metafile

Character. A symbol used in printing. For example, a letter of the alphabet, a numeral, a punctuation, or any other symbol that represents information.

Character set. (1) A finite set of different characters that is considered complete for some purpose. (2) The object format data set component that contains the character graphics and their descriptions. (3) The source definition that specifies included character groups and printing properties.

Characters per inch. The number of characters printed horizontally within an inch across a page.

CL. The set of all commands with which a user requests system functions (AS/400).

CMY. Cyan, Magenta, Yellow (colors)

Coded font. (1) A font component that associates a code page and font character set. (2) A font that is fully described in terms of typeface, point size, weight, width, and attributes.

Code page. (1) A particular assignment of hexadecimal identifiers to graphic characters. (2) (ADVPRINT) A font file that associates code points and graphic character identifiers.

Code point. A 1-byte code representing one of 256 potential characters.

Composed-text data. Text data and text-control information that dictates the format, placement, and appearance of the data to be printed.

Composite document. A document that contains graphics or images as well as text. See 1.2, "Terms with Special Meanings" on page 2.

Composition. The act or result of formatting a document.

Computer-aided design/computer-aided manufacturing. An application in which devices such as personal computers can be used to design and develop products such as circuit boards, machine hardware, and other mechanical and electrical parts.

Control language. see CL

CPF. Control Program Facility (AS/400)

D

Database. (1) A set of data, part or the whole of another set of data that consists of at least one file, and that is sufficient for a given purpose or for a given data-processing system. (I) (A) (2) A collection of data fundamental to a system. (A)

Data object. Part of a data stream; see 1.2, "Terms with Special Meanings" on page 2.

Data Stream. For the purposes of this book, a data stream is a format definition of data which can be used to define a composite document. Therefore, AFPDS and PostScript are data streams; but TIF is not a data stream because it cannot describe all objects in a composite document. TIF is therefore defined as a *file format*, and TIF files are *data objects* that can form part of a composite data stream.

For a full discussion of these points, see 1.2, "Terms with Special Meanings" on page 2.

DB. database

DCA. Document Content Architecture

DCF. Document Composition Facility.

DFU. Data File Utility (AS/400)

DIA. Document Interchange Architecture

DIF. Display Information Facility

DisplayWrite. An IBM licensed program that provides word-processing capabilities for a number of printers.

DLF. Document Library Facility, licensed program which allows the SCRIPT/VS formatter to operate in batch environments and MVS and VSE.

Document. In word processing, a collection of information that pertains to a particular subject or related subjects.

Document Composition Facility. An IBM

licensed program that provides text formatting for a number of printers.

DOS. Disk Operating System

DPE. Decompression Performance Enhancement (IBM 3900)

DTD. SGML Document Type Definition

DTP. Desk Top Publishing

Duplex. Usually wrongly used to mean a mode of copying or printing on both sides of a sheet ("Duplex Printing"). "Duplex" is properly the process of formatting a data stream such that pages are alternately formatted into different layouts to suit printing on the two faces of a sheet of paper (to keep larger margins at the bound edge, page numbers out, etc.).

DVI. Version independant data stream of TEX

DW. DisplayWrite

DW/370. DisplayWrite /370

DW5/2. DisplayWrite 5/2

DW5/2C. DisplayWrite 5/2 Composer

E

EAN. European article number (barcode format)

EBCDIC. Extended Binary Coded Decimal Interchange Code; standardized format and collating sequence for internal representation of characters, used by most IBM host systems.

Editing. the processes associated with creating and amending the structure or content of documents.

Electronic form. A collection of constant data that is electronically composed in the host processor and can be merged electronically with variable data on a sheet during printing. See preprinted form. See also overlay.

Enterprise. Within the context of this document, a business organization which is dispersed and divided into multiple functional and geographical units.

EPS. Encapsulated PostScript

External formatting. Controls for the placement of data on the page that are imbedded outside the actual application program.

F

FD:OCA. Formatted Data Object Content Architecture

FFT. Final Form Text

FFT:DCA. Final Form Text Document Content Architecture

FLSF. Font Library Services Facility.

FOCA. Font Object Content Architecture

Font. (1) An assortment of characters of a given size and type style. (2) (SAA) A particular style of type (for example, Bodini or Times Roman) that contains definitions of character sets, marker sets, and pattern sets.

Font character set. A font file that contains the raster patterns, identifiers, and descriptions of characters.

Font ID. A number that identifies the character style and size for certain printers.

Font Library Services Facility. An IBM licensed program used to maintain font libraries.

Font object. A member of a font library.

Form. (1) The paper on which output data is printed by a line printer or a page printer. (2) A physical sheet of paper. (3) See electronic form, preprinted form.

Format. (1) A specified arrangement of such things as characters, fields, and lines, usually used for displays, printouts, or files. (2) To arrange such things as characters, fields, and lines.

Formatting. The process of determining the appearance of the content of a document on a presentation medium.

FORMDEF. Form Definition. See also form definition

Form definition. A resource that defines the characteristics of the form which include overlays to be used (if any), text suppression, the position of page data on the form, and the number and modifications of a page. Contrast with page definition.
A form definition is referenced in the print job an interpreted by PSF. Form definitions may be created with PMF, PPFA or complementary software.

Form type. A 10-character identifier, assigned by the user, that identifies each type of form used for printed output.

Forms flash. On the 3800 Printing Subsystem, a means of printing an overlay using a negative plate projected on a form.

G

GDDM. Graphical Data Display Manager.

Generalized Markup Language. An IBM licensed program that identifies the parts of a source document without respect to a particular processing system.

GDF. A Graphics Data Format file used extensively within GDDM and GDQF. It contains graphics data in a device-independent format that can be converted to a suitable format for several different types of output devices. The GDF/ADMGDF is defined by the Graphical Data Display Manager (GDDM).

GDQF. Graphic Display and Query Facility

GGCA. Geometric Graphics Content Content Architecture

GIF. Graphics Interchange Format

GKS. Graphics Kernel System

GL. Graphics Language.

GML. Generalized Markup Language.

Glyph. A glyph is a graphic shape which is one of the representations of a character or symbol. A character code defines a given character, given a code page. That character has one or more glyphs which can represent it. A typeface will have a representation of one of those glyphs.

GOCA. Graphics Object Content Architecture

Graphic character-set ID. A 5-digit registered identifier used to specify a graphic character set. The graphic character-set ID is the first part of the QCHRID system value or the CHRID parameter value. See also code-page ID.

Graphical Data Display Manager. Is a host-based product providing graphics manipulation and display services for applications.

H

HPGL. Hewlett-Packard Graphics Language

HPGL/2. Hewlett-Packard Graphics Language contains extensions over the original language

HPPCL. Hewlett-Packard Printer Command Language (PCL3, PCL4, PCL5)

I

IBMGL. IBM Graphics Language

IBM-GL. IBM Graphics Language

ICU. Interactive Chart Utility (GDDM)

IDU. PC/IDU, Image Data Utility

IGES. Initial Graphics Exchange Specification

IHF. Image Handling Facility.

IIA. Information Interchange Architecture

Image. (1) Pictorial information that is specified in terms of the dots (pixels) of which is made up.

Image Handling Facility. An IBM licensed program that prepares black and white or color images to be used as page segments in documents.

IMD. Interactive Map Definition

IMDS. image data stream

Impact printer. (1) A device in which printing results from mechanical impacts. (I) (A) (2) Contrast with non-impact printer.

Intelligent printer data stream. (1) An all-points-addressable data stream that allows users to position text, images, and graphics at any defined point on a printed page. (2) (Graphics) A structured-field data stream for managing and controlling printer processes, allowing both data and controls to be sent to the printer.

Interchange. the process of providing a person or device with a document by means of data interchange or exchange of storage media.

IOCA. Image Object Content Architecture

IS. Interchange Sets (MO:DCA)

ISE. Image Symbol Editor, part of GDDM

ISGL. Industry Standard Graphics Language

ISO. International Organization for Standardization

IVU. Image View Utility (GDDM)

J

JPEG. Joint Photographic Experts Group (compression algorithm)

K

Kanji. A character set of symbols used in Japanese ideographic alphabet.

L

LATEX. Language TEX

Line data. Information that enables a line printer to print one line at a time. Line data is usually characterized by carriage control characters and table reference characters. See composed-text data.

Line printer. A device that prints a line of characters as a unit. Contrast with page printer.

Lines per inch. The number of characters that can be printed vertically within an inch.

Logical page. The boundary for determining the limits of printing on a physical page as determined by software commands. Contrast with physical page.

Lpi. See lines per inch.

LU6.2. SNA Logical Unit Type 6.2; communications protocol definition for peer-level communication between intelligent devices over an SNA network.

LZW. Lempel-Ziv Welch (compression algorithm)

M

Mixed-pitch font. A font that simulates a typographic font. The characters are in a limited set of pitches; for example, 10-pitch, 12-pitch, and 15-pitch.

MMR. IBM Modified Modifed Read (De-, Compression Algorithm)

MO:DCA. Mixed Object Document Content Architecture

MO:DCA-L. Mixed Object Document Content Architecture for Library

MO:DCA-P. Mixed Object Document Content Architecture for Presentation

MO:DCA-R. Mixed Object Document Content Architecture for Resources

Multiple up. The printing of more than one page on a single surface of a sheet of paper.

N

Non-impact printer. (1) A device in which printing is not the result of mechanical impacts; for example, thermal printers, electrostatic printers, photographic printers. (I) (A) (2) Contrast with impact printer.

O

OCR. Optical Character Recognition

ODA. Office Document Architecture

Offset stacking. A function that allows the printed output pages to be offset for easy separation of the print jobs.

OGL. Overlay Generation Language.

OIS. OS/2 Image Support

Overlay. A collection of predefined data such as lines, shading, text, boxes, or logos that can be merged with variable data on a page while printing. See also electronic form, preprinted form.

Overlay Generation Language. An IBM licensed program used to create electronic forms.

P

PAGEDEF. Page Definition. See also page definition

Page definition. An object containing a set of formatting controls for printing logical pages of data. It includes controls for number of lines per printed sheet, font selection, print direction, and mapping individual fields in the data to positions on the printed sheets. Contrast with form definition.
A page definition is referenced in the print job an interpreted by PSF. Page definitions may be created with PMF, PPFA or complementary software.

Page printer. A device that prints one page as a unit. (I) (A) Contrast with line printer.

Page Printer Formatting Aid. An IBM licensed program that allows for creation and storage of form definitions and page definitions—resource objects for print-job management.

Page segment. An object containing composed text and images, prepared before formatting and included during printing.

PC. Personal Computer (also usually includes PS/2)

PCL. Printer Command Language

PCX. Paintbrush PCX File

Pel. A *picture element (pixel* or *pel)* is the smallest element of a displayable or printable surface that can be independently assigned color and intensity.

PDL. Page Description Language

PFD. Print Format Definition (AS/400)

PFU. Print Format Utility (AS/400)

PGF. Presentation Graphics Feature

Phototypesetting. The process of producing high quality text by means of a photographic process.

Physical page. The side of a sheet of paper that is to be printed on. Contrast with logical page.

Picture element. An element of a raster pattern about which a toned area on the photoconductor might appear. See also raster pattern and pel.

PIF. Picture Interchange File

Pixel. see pel

Plotter. An output unit that presents data in the form of a two-dimensional graphic representation. (I) (A)

PMF. Print Management Facility.

Post-processing option. A hardware device that attaches to the output side of a printer; for example, an envelope stuffer, binder, or stapler.

PPDS. Personal Printer Data stream

PPFA. Page Printer Formatting Aid.

Preprinted form. A sheet of paper containing a preprinted design of constant data. Variable data can be merged on such a form. See electronic form.

Pre-processing option. A hardware device that attaches to the input side of a printer; for example, a paper roll feed or multiple input bins.

Presentation. the operation of rendering a document in a form perceptible to a human.

PRF. Print Request Facility, directs output to any printer. Is part of the SAA PrintManager.

Print Descriptor. Print Descriptor is collection of data describing the characteristics of the page, PSF resources, GDDM tokens and the routing information for SAA PrintManager.

Print Services Access Facility. A menu-driven, IBM licensed program that allows the user to select print parameters for page printers controlled by PSF.

Print Services Facility. An IBM licensed program that produces printer commands from the data sent to it.

Printer driver. A program that passes commands and resources with a data stream from the system spool to tell the printer how to print the page.

Printer Management Facility. An IBM licensed program that creates fonts, page segments, page definitions, and form definitions.

PS/2. Personal System/2 (See PC)

PSAF. Print Services Access Facility.

PSEG. Page Segment (rasterized file from GDDM)

PSF. Print Services Facility.

PSF/2. Print Services Facility under OS/2

PTOCA. Presentation-Text Object Content Architecture

Q

R

Raster graphics. (1) Computer graphics in which a display image is composed of an array of picture elements (pels) arranged in rows and columns. (I) (A) (2) Contrast with vector graphics.

Remote PrintManager. A personal computer product that allows selected font data, overlays, and page segments that are present in advanced function printing data streams to be available to a locally attached IBM page printer.

Resource library. (1) A collection of related files. (2) A place to store resources, such as form definitions, page definitions, page segments, fonts, and overlays.

RFT. Revisable Form Text

RFT:DCA. Revisable Form Text Document Content Architecture

RFTXT. Revisable Form Text Documents

RGB. Red, Green, Blue (colors)

RGCA. Raster Graphics Content Content Architecture

RSU. Resource Service Utility (AS/400)

RTF. Rich Text Format

Rule. A solid or patterned line of any weight, extending horizontally or vertically across a column or row.

S

SAA. System Application Architecture

Scan line. (1) One horizontal sweep of the laser beam. (2) A single row of pels.

SCRIPT. A formatting program used by Document Composition Facility for processing text.

SCS. SNA Character Stream

SGML. Standard Generalized Markup Language

Simplex. Pertaining to formatting for print on one face of the paper. Cf. duplex.

SMFF. Script Mathematical Formula Formatter

SPDL. Standard Page Description Language

Systems Application Architecture. Is a set of defined rules, guidelines, interfaces and protocols for application and system design, intended to facilitate cross-system consistency and application portability.

T

Text. A graphic representation of information on an output medium. Text consists of alphanumeric characters and symbols arranged in paragraphs, tables, columns, or other shapes.

Text-formatting program. A program that determines the manner in which data will be placed on a page.

TIF. TIF stands for Tagged Image Format. On some systems it appears as *TIFF*, standing for Tagged Image Format File. The format is more fully described in section 10.3, "Tagged Image File Format (TIF)" on page 63.

Typeset. (1) To arrange the type on a page for printing. (2) Pertaining to material that has been set in type.

Typographic font. A typeface originally designed for typesetting systems. Contrast with mixed-pitch font, uniformly spaced font.

U

Unbounded-box format. An organization of character graphics and information used by advanced-function-printing programs for printing on the IBM 3800 Models 3, 6, and 8. Character boxes containing each character graphic do require character-positioning information in the form of untoned pels in the character boxes. In addition, a separate character set is required for each combination of character rotation and text orientation. Fonts in unbounded-box format usually require more storage than fonts in bounded-box format. Contrast with bounded-box format.

Uniformly spaced font. A font in which the characters have the same character increment. Contrast with typographic spaced font.

UPC. Universal Product Code (barcode format)

V

Vector. In computer graphics, a directed line segment.

Vector graphics. (1) Computer graphics in which display images are generated from display commands and coordinate data. (I) (A) (2) Contrast with raster graphics.

W

Word processing. The entry, modification, formatting, display, and printing of text on personal computers, microprocessors, and stand-alone word processors.

Work station. (1) A configuration of input/output equipment at which an operator works. (T) (2) A terminal or microcomputer, usually one that is connected to a mainframe or to a network, at which a user can perform applications.

WYSIWYG. What You See Is What You Get

X

XICS. XEROX Integrated Composition System

Y

Z

Zoom. The progressive scaling of an entire display image in order to give the visual impression of movement of all or part of a display group toward or away from an observer. (I) (A)

Index

Numerics

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

GG24-3938-00

Print and View Data Streams

GG24-3938-00

Printed in U.S.A. on Recycled Paper

IBM®