Volume 2, Number 10 November 1984

IBM Personal Computer Seminar Proceedings

The Publication for Independent Developers
of Products
for IBM Personal Computers

Published by International Business Machines Corporation
Entry Systems Division

Contents

Introduction and Welcome

These are the Proceedings of the IBM Personal Computer Seminar, designed for independent developers of products for IBM Personal Computers. The purpose of these Proceedings is to aid you in your development efforts by providing relevant information about new product announcements and enhancements to existing products. This issue is prepared in conjunction with this seminar. The Proceedings of future siminars for the IBM Personal Computers also will be published and will cover topics presented at those seminars.

Throughout these Proceedings, the term Personal Computer and the term family of IBM Personal Computers address the IBM Personal Computer, the IBM Personal Computer XT, the IBM PC*jr*, the IBM Portable Personal Computer, and the IBM Personal Computer AT.

Purpose

What is our purpose in putting out a publication such as this? It is quite simple.

The IBM Personal Computer family is a resounding success. We've had a lot of help in achieving this success, and much of it came from the independent developers.

As you proceed with your development, do you at times wish for some bit of information or direction which would make the job easier? Information which IBM can provide? This is the type of information we want to make available to you.

Since we want to be assured of giving you the information you need, we ask you to complete the questionnaire which appears at the end of these Proceedings. Your response to this questionnaire will be taken into account in preparing the content of future issues, as well as the content of seminars we will present at microcomputer industry trade shows.

Topics

The following list gives a general indication of the topics we plan to cover in future seminars and include in the IBM Personal Computer Seminar proceedings:

- Information exchange forum — letters to the editor format
- Development tools — languages, database offerings
- Compatibility issues
- New devices — capacities and speeds
- System capacities — disk and memory
- Enhancements in maintenance releases
- Tips and techniques
- New system software
- Hardware design parameters
- Tips on organizing and writing documents for clear and easy reading
- Changes to terms and conditions

IBM PC Professional Graphics Software

Introduction

On September 10, 1984 IBM announced a number of products intended for use in the engineering and scientific communities. Among the new products announced were five professional graphics software development tools, a high resolution controller and display, an IBM PC Professional FORTRAN, and hardware and software to address the control and instrumentation needs of both the industrial and educational engineering/scientific environments.

This document focuses on the five graphics software products and the IBM Professional Graphics Controller/Display. The first section gives a product summary and overview. More detailed descriptions of three of the products are contained in the second section and finally, sample program listings, written in IBM Professional FORTRAN, are available for your information.

IBM Professional Graphics Product Summary

The five graphics software development tools announced for the IBM family of Personal Computers are:

The IBM Graphics Development Toolkit

This is the foundation product for all of the IBM PC Professional Graphics Software products. Included in the Toolkit is a Virtual Device Interface (VDI) which follows the proposed ANSI X3H33 definition.

The IBM Graphical Kernel System

The IBM Graphical Kernel System (GKS) is an implementation of the proposed ISO/ANSI GKS level mb standard.

The IBM Plotting System Library

The IBM Plotting System Library (PSL) is a VDI based product intended to provide a tool for the development of a variety of presentation level graphics.

The IBM Graphical File System

The IBM Graphical File System (GFS) is a tool which follows the proposed ANSI Virtual Device Metafile (VDM) for the storing, retrieving and the manipulation of graphic images. The product provides a programming and an interactive interface for application developers.

The IBM Graphics Terminal Emulator

The IBM Graphics Terminal Emulator provides the means to interface with host systems, allowing access to sophisticated graphics software products from an IBM Personal Computer emulating either the Tektronix* 4010 Series terminals or the Lear Siegler ADM 3A** terminal.

The IBM Professional Graphics Controller/ Display

The IBM Professional Graphics Controller and display is a high resolution controller/display with many built in graphic functions. The controller/ display will be useful in many areas including computer aided design and manufacturing, CAD/ CAM, and other graphics applications requiring high resolution graphic images. The new IBM Personal Computer AT complements the IBM Professional Graphics Controller/Display.

IBM Professional Graphics Overview

The Graphics Development Toolkit

The Graphics Development Toolkit (VDI) provides the foundation for all of the new IBM PC Professional Graphics software products. It is intended for use in graphics applications for the PC. It can be used by itself or in combination with other IBM PC Professional Graphics products.

This VDI product is intended for use in creating simple and complex graphics images and at the same time, VDI can provide a standard device interface. The application programmer is free to design a graphics solution without regard to which of the many input/output graphics devices end users may use on their particular system. A brief overview of the Toolkit follows.

The VDI Controller

The Virtual Device Interface Controller is a common language interface which allows device-independent software and device-dependent drivers to communicate. VDI also provides both basic and Generalized Drawing Primitives (GDP).

The Device Drivers

The drivers communicate directly with the VDI Controller and the graphics devices. Each specific device will have its own driver and the driver will translate all the information exchanges between the devi-ce and application program.

Linkable Libraries

VDI has a set of linkable libraries for graphics and text functions. The VDI functions are grouped into eight major functional areas:

Workstation Control Functions
Paging Functions
Pel Functions
Cursor Control Functions
General Graphic Functions
Graphic Functions and Attributes
Text Functions
Input Functions

Reference Material

Reference material containing specific language syntax for each of the supported functions is included in the documentation. The address of where and how to obtain information about redistribution and how to write your own drivers to interface with the VDI is:

IBM Corporation
Graphics Development Toolkit
P. O. Box 1328-A
Boca Raton, FL 22432

Drawing Primitives

Drawing primitives include polyline, polymarker, and text models. The polyline primitive draws vectors between sequences of end points. The polymarker places a marker symbol at each point in the array and the text primitive displays text strings at any position with any orientation. Three types of text models are supported, alpha, graphic and cursor.

The VDI also supports raster devices, fill and cell array primitives and raster operation or Pel functions. The Pel functions move one or more pels and can provide for animation and image generation. The VDI also provides for inquiry operations to aid programmers.

The Graphics Development Toolkit provides a set of device drivers for a variety of IBM devices including the PC*jr* Joystick, several displays, printers and plotters.

Language interfaces include IBM PC BASIC Compiler 1.00, IBM FORTRAN Compiler 2.0, IBM PC Professional FORTRAN, Pascal Compiler 2.00, Macro Assembler and Lattice C 2.0 developed by Lattice, Inc.

The Graphical Kernel System

The Graphical Kernel System (GKS) is implementation of the proposed ANSI standard level mb with full 2b segmentation features. GKS segmentation is a collection of graphics primitives that are dealt with as a single unit. Graphics functions including transformation, scaling, and highlighting among others can be used in conjunction with the respective segment.

The IBM PC Graphical Kernel System product like the VDI product is intended for use by programmers and application developers. GKS is designed to provide portability among systems that support the proposed standard.

Five input functions locator, string, pick, choice and valuator are supported. Also supported are 2D graphics primitives. Transformation between world, normalized device coordinate and device coordinate systems are supported.

Language interfaces with the exception of Pascal and Assembler are same as those supported by the Toolkit.

The IBM Plotting System Library

The Plotting System Library (PSL) provides a set of the tools to aid in development of high quality graphics. Using the foundation provided by the VDI, the Plotting System Library allows the use of a large variety of I/O devices but without concern for which device will actually be used with the application.

PSL is a collection of 2D graphics subroutines including chart types, chart attributes, graphic and text annotation, inquiry and other functions. PSL is intended for use by experienced and novice users.

The language interfaces supported in the GKS environment are also supported by the Plotting System Library.

The Graphical File System

The Graphical File System converts and stores graphic information in accordance with the proposed ANSI Metafile Standard. Through the Metafile, the user can retrieve graphic information to re-create an image or combine information from more than one source to form an entirely new image.

There are two methods of using the Graphical File System; one is to interact with the image information through a program interface and the other is through an interactive session at the PC keyboard. The product provides language interfaces identical to GKS and PSL.

The Graphics Terminal Emulator

The Graphics Terminal Emulator provides the host system interface to allow IBM PC users to interact with sophisticated graphics applications by emulating two terminals common to this environment. The terminals emulated are the Tektronix* 4010 series and the Lear Siegler ADM-3A.**

The Graphics Terminal Emulator uses the VDI to provide access to a wide variety of devices on the PC and can operate with the IBM System/370 and non-IBM systems. An icon driven user interface is provided for speed and ease of use.

The user can upload and download ASCII data streams and the data can be examined while continuing to operate in the emulator mode. Communications between host and emulator are conducted through an ASCII RS-232C communications port.

IBM Professional Graphics Requirements

The Professional Graphics Products have minimum system requirements and the publications associated with each of the products should be consulted for specific information. There are, however, some general requirement comments to make regarding the graphics products.

The Professional Graphics Controller/Display can be used in all IBM PC models except the PC*jr.* The Controller/Display requires the PC expansion unit on the IBM PC and IBM Portable PC. The controller requires two adjacent slots because of the card configuration.

The general minimum product requirements are:

128KB of storage for the Toolkit
256KB for the other software products
DOS 2.1 or higher
A language compiler
1-360KB diskette drive for the Toolkit/Emulator
2-360KB diskettes, a 1.2MB diskette drive or a hard file for all others
For the Professional FORTRAN, a 8087 or 80287 IBM Math Co-Processor.

The Graphical Kernel System, the Plotting System Library, the Graphical File System, and the Graphics Terminal Emulator each come with a copy of the VDI Controller and Device Drivers elements of the Toolkit. Any applications developed using these products for distribution require a redistribution license from IBM. The address for information on this topic is shown in the Reference Material section.

The following articles provide greater details about three of the IBM Professional Graphics Products; the Graphical Kernel System, the Plotting System Library and the Graphical File System.

* Tektronix is a trademark of Tektronix, Inc.
** Lear Siegler ADM-3A is a trademark of Lear Siegler, Inc.

IBM PC Graphical Kernel System

Introduction

The IBM Graphical Kernel System (GKS) is a powerful and easy to use tool for programmers and application developers. It provides a set of functions that can be used by the majority of applications that produce computer generated pictures.

The IBM Graphical Kernel System (GKS) is based on proposed ISO/ANSI standards to aid graphics application programmers in understanding and using graphics methods, and to guide device manufacturers on useful graphics capabilities.

The GKS was designed to provide program portability between computer systems, a task accomplished by achieving source code compatibility that provides a consistent interface in high level languages.

By enabling programmers to work to constant standards, programs conforming to GKS specifications can be combined with new and more sophisticated programs. Any program written to GKS standards will work on any system supporting this standard.

The Graphical Kernel System is built upon the VDI and benefit from its versatility. It can use the built in attributes of a given device, or it can emulate functions and provide high level support to less sophisticated devices.

Highlights

Provides portability of graphics application between computer systems that support the GKS standard. The level of portability for executing a graphics application is dependent on each computers functional capabilities and implementation.

- Provides device independence to graphics application.
- ANSI level mb implementation with full 2B segmentation features of the GKS standard.
- Supports segmentation functions. Segment is a collection of graphics primitives that can be dealt with as a unit through a range of graphics manipulations, including transformation, scaling, highlighting, visibility and detectability.
- Rich set of inquiry functions.
- Five different classes of input functions: locator, string, pick, choice, and valuator.
- Support for 2D graphics primitives.
- Provide the following language bindings:

  FORTRAN 2.0
  Professional FORTRAN
  Lattice C
  Basic compiler

- Supports transformation between world, normalized device coordinate and device coordinate systems.

Workstations

GKS supports four categories of workstations:

- OUTPUT (Output only). For example: printers.
- INPUT (Input only). For example: joystick.
- OUTIN (Output and input). For example: display with keyboard can be used to output graphical image and input key strokes.
- WISS (Workstation independent segment storage). WISS is a virtual device which allows segments to be stored independent of any particular physical workstations.

GKS allows a maximum of three workstations to be opened at any one time: one OUTPUT or OUTIN, one INPUT, and WISS.

GKS Operating States

During processing, GKS is always in one of the following five operating states:

1. Kernel System closed
2. Kernel System open
3. At least one workstation open
4. At least one workstation closed
5. Segment open

Each GKS routine requires that the system be in a certain operating state. If you make a call in the wrong operating state, you will receive a 'Kernel System not in proper state' error message.

GKS Routines

The GKS routines are separated into nine categories that group the routines according to the task each one performs, including:

- Control routines. Routines that affect the state of the system or of individual workstations.
- Output routines. Routines that display basic graphic images, for example: polylines, polymarkers, fill areas, text, ...etc.
- Attribute routines. Routines that modify the appearance of graphic primitives mentioned above.
- Transformation routines. Routines that translate primitives between coordinate planes.
- Segment routines. Routines that create, delete, and manipulate segments.
- Input routines. Routines that make interactive graphics applications easy.
- Inquiry routines. Routines that return data to the application, including information about workstations, logical input devices, and current attribute settings.
- Utility routines. Routines that pack or unpack data records and compose segment transformation matrixes.
- Error handling. Routines that assist the application program in logging and handling errors.

Control Routines

Control routine affect the state of the system or the state of the workstation.

Control routines include: initialize and terminate the Kernel System, open, activate, deactivate and close workstation, clear workstation, update workstation, redraw all segments on workstation, and escape routine.

The escape routine allows you to issue commands directly to the device, to take advantage of any non-standard features available on your particular graphics device.

Output Routines

Routines that display basic graphic images, including: polylines, polymarkers, fill areas, text, and generalized drawing primitives. Generalized drawing primitives include: circles, arcs, pie slices, and bars.

Attribute Routines

Routines that modify the appearance of graphic primitives mentioned above.

For polylines, you can set the following attributes:

- Polyline color index.
- Polyline width scale factor.
- Polyline type, at least 6 line types are guaranteed.

For polymarkers, you can set the following attributes:

- Polymarker color index.
- Polymarker size scale factor.
- Polymarker type, at least 6 marker types are guaranteed.

For fill areas, you can set the following attributes:

- Fill area color index.
- Fill area interior styles, which include: hollow, solid, pattern, and hatch
- Fill area style index, which is a index into the pattern table or hatch table. At least 6 hatch styles are guaranteed.

For text, you can set the following attributes:

- Text color index.
- Text font and precision. Precision includes: string precision and character precision.
- Character height.
- Character up vector, the direction of the text string.
- Text alignment, includes 3 possible horizontal justifications, and 3 possible vertical justifications.

GKS does not provide attributes explicitly for generalized drawing primitives. Each generalized drawing primitive assume the current attributes of the GKS primitive it most closely resembles. Arcs use current polyline attributes, Bars, pie slices, and circles use fill area attributes.

The number of available choices for an attribute are sometimes determined by the device capability. For example, the number of text fonts available is determined by how many hardware fonts the device supports and the number of polyline colors available is determined by how many colors the device supports.

Transformation Routines

There are three coordinate systems in GKS. The GKS transformation routines perform mapping between the three coordinate planes.

The world coordinate (WC) plane is a user defined cartesian coordinate system. You build your graphics image in WC coordinates. You define the range of WC with the set window routine.

The Normalized device coordinate (NDC) plane is a standardized, virtual plane that provides a uniform coordinate system for all workstations. You define the range of NDC with the set viewport routine and set workstation window.

The device coordinate (DC) plane is the portion of the device surface that will be used to output graphics. You set the range of DC with the set workstation viewport routine.

Normalization transformation maps WC to NDC, and workstation transformation maps NDC to DC.

Eight normalization transformation numbers can be defined at one time in GKS. Select transformation number routine is used to select the effective transformation number.

By default, the normalization transformation number 0 is used to map world coordinate (0.0, 1.0) X (0.0, 1.0) to NDC (0.0, 1.0) X (0.0, 1.0), and the default workstation transformation maps NDC (0.0, 1.0) X (0.0, 1.0) to DC using the largest square that will fit on the device surface.

Segment Routines

When primitives are grouped together in a segment, you can perform operations on them as a single object.

To create a segment, you call create segment routine, then calling the output routines to create the primitives for the segment. The close segment routine is called to define the end of the current segment.

When a segment is created, it is automatically stored in all output workstations that are active. To store a segment into a workstation, the associate segment with workstation routine can be used.

To delete a segment from all workstations, the delete segment routine is used. To delete a segment from certain workstations only, the delete segment from workstation routine is used.

Each segment has attributes associated with it. Each attribute is set by a segment routine. These attributes include:

- Visibility
- Detectability
- Highlighting
- Segment priority
- Pick identifier

As long as a segment is created, it can be transformed by the set segment transformation routine. These transformations are accomplished by a 2X3 transformation matrix, which is calculated by the evaluate segment transformation matrix routine. The routine accepts fix point, shift vector, rotation angle, and scaling factors as input parameters, and produce a transformation matrix as an output parameter.

Input Routines

GKS supports 5 input classes for interactive graphics applications, which are locator, choice, string, pick and valuator.

Locator - Used to select a position on the display surface by moving a graphics input cursor to the desired position. The value returned by locator is a point in world coordinates, together with a normalization transformation number.

Choice - Used by the user to make a choice on a choice device. The keyboard function keys are simulated as a choice device. The choice function returns a nonnegative integer value that represents a selection from a number of choices. Usually a choice device asks an operator to choose among fixed alternatives by pressing a button or function key.

String - Used to enter a character string into the program. Returns a character string, typically from a keyboard.

Pick - Used to pick a segment on the display surface by moving the graphics input cursor over it. The pick function returns a segment name, a pick identifier and the status.

Valuator - Used to enter a real number value by setting a valuator device, for example, turning a dial to the position which represents the value you want.

Each input function operates in one of two modes, sample mode or request mode.

In request mode, the program operation is held until the operator responds with a request mode trigger. For example, when request locator is invoked, program processing is halted until the user moves the cursor to the desired position on the screen and presses the enter key on the keyboard or the corresponding buttons on the joystick as a trigger to terminate request mode.

In sample mode, the value returned by the logical input device is the current measure of the physical workstation. For example, the sample locator returns the current position of the graphics input cursor at the time the routine is called.

Inquiry Routines

GKS provides an extensive set of inquiry routines that return information to application programs about the current operating state, workstation information, current setting of primitive attributes, segment attributes, device capability...etc.

Utility Routines

GKS provides utility routines for your convenience in computing transformation matrixes and handling packed data records.

Error Handling

GKS provides error handling and error logging routines to react to error situations.

For each GKS routine, a finite number of error situation is defined, and can be classified as following:

Class 1 - Errors resulting in a precisely defined reaction. For example, viewport rectangle set is invalid, GKS reacts by logging an error message in the error file of the application program, and ignore the function call.

Class 2 - Errors resulting in an attempt to save the results of previous operations. For example, out of memory errors or failure to open a workstation, GKS reacts by displaying an error message on the console, and a transfer of control to the operating system console.

Class 3 - Errors that can cause unpredictable results including the loss of information or control. For example, hardware failure.

Prerequisites

DOS version 2.1 or later

Fortran 2.0, Professional FORTRAN, Lattice C V2.0 by Lattice Corporation, or another compatible C compiler.

An IBM Personal Computer, IBM Personal Computer XT, IBM Portable Personal Computer, IBM Personal Computer AT with a minimum system configuration of the following:

256 K memory
2 double sided diskette drives
Graphics display

Recommended configuration:

512 K memory
8087 or 80287 Math Co-processor
IBM Fixed disk and double-sided diskette drive
Hardcopy device and input device

Packaging

The Graphical Kernel System is packaged on four double sided diskettes with the Graphical Kernel System manuals. Device drivers and selected Graphics Development Toolkit modules are included on two additional diskettes.

Publication

The following publications will be available with the software:

- Programmers Guide
- Language Bindings, Volumes 1, 2, and 3

IBM Personal Computer Graphical File System

Introduction

This article provides an overview of the IBM Graphical File System. The key features of the product are here. More complete information is contained in the publication, IBM Graphical File System Programmer's/User's Guide.

This article covers the following topics:

- Metafile – Concepts, Structure , and Benefits
- Encoding
- Interfaces
- Pre-Requisites, Packaging, and Installation

Metafile - Concepts, Structure, and Benefits

Concepts

The Graphical File System is an implementation of the Virtual Device Metafile (VDM) is a standard way to save pictures on a disk or diskette.

Structure

The overall structure of a metafile consists of header and trailer data, and the representation of one or more pictures. Each picture consists of header and trailer data, the data specific to the picture itself, and a set of variable length metafile elements or instructions made up of operation codes and various parameters. These instructions are processed by a program known as a metafile interpreter when the picture is to be displayed.

The structure and definition of the metafile elements is consistent with the proposed ANSI X3H33 standard. The metafile is created by an appropriate device driver operating under the Virtual Device Interface (VDI) which translates the graphics commands issued by an application program to the metafile encoding.

When recreating a picture, a metafile interpreter program will read the file, interpret the elements and reissue the VDI commands for execution by the output device driver. The output can be directed to a printer, plotter or display regardless of the type of device originally used for displaying the picture.

The output can also be directed to a metafile to save as a new picture. Thus a metafile containing separate pictures of three ships could be combined to show all three ships in one picture and the resulting picture saved.

Benefits

Some of the benefits of the Graphical File System, VDM, are described here. Certainly many more will be realized by individual developers as they use the product.

Portability / Interchangeability

Since the pictures are stored in an application and device independent manner, they can be ported to and shared with other computers and installations without recalculation; all that is needed is a metafile interpreter.

Creating New Pictures from Old Ones

Images can be composed from existing metafile images and some rudimentary editing may be performed during the composition process.

Picture Archiving

Fundamental to the VDM notion is the ability to conveniently archive graphic images on commonly available storage media.

Encoding

A metafile consists of metafile elements, which in turn are subdivided into six classes:

Descriptor Elements
Control Elements
Picture Elements
Graphical Elements
Attribute Elements
Escape Elements

Depending on the number of parameters, the elements are characterized as short or long. The first 11 bits define the operation code with bits 5 through 11 defining the operation ID within a class with bits 12 through 15 specifying the class.

Basic Parameter Types

The two basic parameter type are string and 16-bit integers. String consist of an encoded list of 1 byte ASCII characters preceeded by a count. In the short command form this is 1 byte and in the long form it is a 16-bit integer. 16-bit integer data can range from −32768 to 32767. All parameters end on the 16-bit boundaries and where necessary, strings are padded to even byte boundaries.

The programming interface allows a programmer to interrogate each element of the picture as it is read from the metafile. Action can be taken depending on the class and ID. Chapter 5 of the IBM Graphical File System Programmer's/User's Guide together with Appendix B provide a detailed description of the operation codes and their meaning.

Interfaces

The Graphical File System offers two user interfaces:

Interactive Interface

Programming Interface

Interactive Interface

The interactive interface allows an end user to communicate with the Graphical File System through an interactive, icon-driven screen display. This interface provides a user friendly environment to:

- Compose and view an image. The user can select the area of the screen for viewing and if desired, combine multiple images on one screen.
- Create a new metafile picture for later viewing.
- Direct the image to an output device such as a plotter or printer.

Other icons permit the user to specify the size of the displayed image and to provide editing by erasing components of an image. For further details on the interactive interface refer to chapter 3 of the Graphical File System Programmer's/User's Guide.

Programming Interface

The programming interface provides functions which allow application programmers to write programs to control and interpret metafiles: in effect creating their own brand of metafile interpreter, or performing some rudimentary editing. One simple example would be to create a presentation from a set of metafile pictures which have been created separately by different applications. The functions are available as a subroutine package to application programmers who create their programs with the aid of the following language compilers:

FORTRAN Version 2.0
PROFESSIONAL FORTRAN
LATTICE C Version 2.0 *
BASIC Compiler

The subroutine calls are resolved when the compiled object modules are linked with the appropriate subroutine library, see below:

Compiler	**Library Name**
FORTRAN Version 2.0	FORMETA.LIB
PROFESSIONAL FORTRAN	PFMETA.LIB
LATTICE C Version 2.0	CMETA.LIB
BASIC Compiler	BASMETA.LIB and MHEAP.OBJ

*LATTICE C is a trademark of Lattice Corporation

The language bindings are described in detail in chapter 4 of the Graphical File System Programmer's/User's Guide.

The 15 functions may be classified as follows:

- Initializing and terminating
- Picture control
- Interpretation functions
- Workstation – Metafile control
- Cutting and Pasting
- Error detection

Initializing and Terminating

- Open Metafile makes a specified metafile available for reading and returns an integer identifier to be used on other function calls.

- Close Metafile is used when interpretation is complete.
- Open Workstation prepares a workstation to receive graphic output, including clearing the display surface. The workstation is specified by using one of the logical workstation names ("DISPLAY", "PLOTTER", etc.), and the function returns an integer identifier (device handle) to be used on subsequent calls.
- Close Workstation is the converse of Open Workstation.

Picture Control

Once the metafile is opened, an individual picture needs to be selected prior to interpretation. This is performed by:

- Open Picture: in which the specific picture is identified by an integer.
- Close Picture: indicates interpretation is complete.

Interpretation Functions

Two classes of functions are available for interpreting:

- A) Interpret Picture: identifies a complete picture for interpretation and also specifies the output device on which it should appear.
- B) Inquire Metafile Item (element) Length: returns the length (in bytes) of the next element in the Metafile preparatory to loading it into a buffer. (It may be recalled that the metafile elements (items) have widely varying lengths)
- Get Metafile Item: reads the next element into the buffer.
- Interpret Metafile Item: interprets the element in the buffer and directs it to the specified output device.

While class A interprets a total picture, class B allows the user to proceed through the metafile element (or item) by element starting at a given picture.

Workstation Metafile Control

This is accomplished by Clear Workstation which:

- Clears CRT devices.
- Displays all pending graphics on plotters and printers.
- Prompts for new paper on plotters; advances to top-of-form on printers.
- If a metafile is specified, a new picture is initiated.

Cutting and Pasting

Pictures are defined in a virtual device coordinate (VDC) space of 0-32767 on the x and y axes. They are portrayed on a viewing surface with (device dependent) dimensions restricted to 0-32767 on each axis. (Thus the CGA is 32767 by 22500) The Graphical File System allows the user to "cut" part of a picture by specifying a rectangular area or window in the VDC space.This is performed by the Set Window function. This "cut" can be "pasted" to any rectangular area on the display surface by means of the Set Viewport function. These functions provide the basic mechanism to superimpose several pictures on one image.

Error Detection

When each function is utilized, an error situation may be determined by using the Inquire Metafile Error function. Appendix A of the IBM Graphical File System Programmer's/User's Guide lists the error codes which may be returned.

Finally, the Inquire Metafile Version function identifies the current version and level of the Graphical File System.

Prerequisites, Packaging, and Installation

Required are:

- DOS Version 2.1 or later.
- 256K memory in an IBM Personal Computer, IBM Personal Computer XT, IBM Portable Personal Computer, or IBM Personal Computer AT.
- For those wishing to use the programming interface:

  – FORTRAN Version 2.0
  – Professional FORTRAN
  – LATTICE C Version 2.0 *
  – BASIC Compiler

*LATTICE C is a trademark of Lattice Corporation

The Graphical File System is packaged on five diskettes, of which two comprise the underlying IBM Virtual Device Interface Controller, Device Drivers, and auxillary programs. The remaining three diskettes contain the following:

- 1) Metafile Interpreter, Font tables, and sample programs.
- 2) FORTRAN VERSION 2.0 and PROFESSIONAL FORTRAN bindings.
- 3) BASIC Compiler and LATTICE C bindings. *

*LATTICE C is a trademark of Lattice Corporation

Installation requires that the DOS procedure files CONFIG.SYS and AUTOEXEC.BAT are set up appropriately.These files are used by DOS during the initialization process and must be in the root directory. Chapter One of the Graphical File System Programmer's/User's guide describes the installation process in detail.

IBM Plotting System Library

Introduction

Purpose of the Plotting System Library

The purpose of the Plotting System Library is to provide a set of 2-dimensional graphics subroutines that allow the programmer to create professional-quality charts. These include area, bar, line, pie, scatter, schedule, step, and text-only charts.

The Plotting System Library provides bindings to several major languages. The Virtual Device Interface (VDI), although transparent to the programmer, is the foundation of this implementation of the system.

Because the Plotting System Library provides a comprehensive set of default values, you can create charts with a minimum number of steps. A built-in chart dimensioning procedure ensures that the sizing and spacing of chart components are consistent and proportional. Multiple charts or chart types can appear on a single display surface.

Most chart attributes can be changed. These include:

- Line Style for line and step charts
- Fill pattern for area, pie and bar charts
- Color
- Text height and font
- Logarithmic axes
- Axis range
- Tick-spacing

The Plotting System Library includes facilities for both graphic and text annotation. In addition, it contains device inquiry capabilities that allow you to use device-independent features. You can also create interactive graphics applications by using input capabilities such as choice, locator and string.

Package Contents

The Plotting System Library is distributed on five diskettes. Two diskettes are common throughout the Professional Graphics Series and contain the VDI controller, supported device drivers and related code. Three diskettes contain the Plotting System Library software. Here are the files contained in the package:

System Files

AUTOEXEC.BAT
CONFIG.SYS
INIT_VDI.EXE
LINK.EXE
VDI.SYS

BASIC

BASPLOT.LIB
PLOT.LIB
PHEAP.ASM
PHEAP.OBJ
BLINE.BAS
BBAR.BAS
BBAR2.BAS
BPIE.BAS
BINPUT.BAS
BTEXT.BAS

Lattice C

CPLOT.LIB
CLINE.C
CBAR.C
CBAR2.C
CPIE.C
CINPUT.C
CTEXT.C

Professional FORTRAN

PFPLOT.LIB
PLOT.LIB
PFLINE.FOR
PFBAR.FOR
PFBAR2.FOR
PFPIE.FOR
PFINPUT.FOR
PFTEXT.FOR

FORTRAN Version 2.00

FORPLOT.LIB
PLOT.LIB
FLINE.FOR
FBAR.FOR
FBAR2.FOR
FPIE.FOR
FINPUT.FOR
FTEXT.FOR

Font Files

FONT101.TBL
FONT102.TBL
FONT103.TBL
FONT104.TBL
FONT105.TBL
FONT106.TBL

VDI Drivers

VDIDY004.SYS
VDIDY006.SYS
VDIDY008.SYS
VDIDY009.SYS
VDIDY00A.SYS
VDIDY00D.SYS
VDIDY00E.SYS
VDIDY00F.SYS
VDIDY010.SYS
VDIGIJOY.SYS
VDIMTFIL.SYS
VDIPLSIX.SYS
VDIPLTWO.SYS
VDIPRCOL.SYS
VDIPRCOM.SYS
VDIPRGRA.SYS

Description

The Plotting System Library contains bindings to the following languages available for the IBM Personal Computer:

IBM BASIC Compiler Version 1.00
IBM Professional FORTRAN by Ryan-McFarland
IBM FORTRAN Version 2.00
Lattice C^{0}

These bindings are supplied in the form of libraries of functions which are invoked within the application program and linked into the executable module with the DOS Link program. The application programmer can thus choose any of the four languages in which to develop his programs.

Because of the size of the libraries and the number of symbols used, the Link program must be Version 2.3 or later.

Contained within each language library are functions grouped into the following categories:

Chart type
Chart attributes
Graphic input classes
Graphic output primitives
Control
Data set definition
Data set attribute definition
Text definition
Inquiry

A detailed list of the functions in each category appears at the end of this article.

Configuration Requirements

The minimum recommended hardware and software configuration for the IBM Plotting System is:

- IBM Personal Computer AT, XT, PC or Portable PC
- 256KB RAM
- Two 360KB diskette drives (or 1.2MB diskette drive for the IBM Personal Computer AT)
- Graphics display and appropriate adapter
- IBM Disk Operating System (DOS) 2.10 or higher (IBM Personal Computer AT requires DOS 3.00)
- One of the following compilers:

  IBM PC BASIC Compiler 1.00 or higher
  IBM PC FORTRAN Compiler 2.00
  IBM PC Professional FORTRAN Compiler 1.00
  Lattice C Compiler[0]

[0]This product was developed using the Lattice C Compiler, version 2.0, developed by Lattice, Inc.

Creating Charts

The Plotting System Library eliminates many tedious programming tasks, without restricting your creativity as an application developer. The following features give you the means to design virtually any chart with speed and ease.

- Chart dimensioning assures consistent and proportional sizing and spacing of chart components
- Chart components and their related attributes can be controlled individually with no programming order dependency.
- Default values provide the most appropriate choice when a chart component or attribute is not specified.

Chart Dimensioning

As a graphics user, you rely on chart components to represent real-world relationships. The accuracy with which related components are sized and positioned determines the effectiveness of the graphics tool. For this reason, the Plotting System takes an interrelated approach to all chart dimensioning. Once you decide the height and width of a chart's display surface, all other dimensions are then set relative to area defined.

Any portion of the physical device's display surface can be specified as the display surface for your chart. Within the display surface is the view area. The view area contains all chart output and is defined as any portion of the display surface.

Several charts can be created on a single display surface by specifying multiple view areas. The position and size of chart components are specified relative to the view area or to the axis system of the chart itself.

Chart Components

The Plotting System provides a flexible structure and an efficient set of building blocks for chart creation. The following chart components and their related attributes are controlled individually and can be defined in any order.

- Data sets are numeric data that can be translated into graphic data such as bars, lines, pies, and so forth.

- Titles are separate character strings associated with a main chart title, chart subtitle, and axis titles. The height and font of each title can be specified.

- Labels include the axis tick labels that define unit divisions for the vertical and horizontal axes, legend labels, and the labels associated with each pie slice within a pie chart. The font and height of labelling text can be defined.

- Graphics primitives are lines, markers, circles, rectangles, arcs, and arrows added to improve the appearance or clarity of a chart.

- Notation is additional character strings that provide remarks about the chart. Notation can appear anywhere within the view area and in any available font and height.

- Frames of specifiable thickness can be set to enclose the view area or a specific chart area.

- Gridding can be generated within the area defined by chart axes. When turned on, grid lines appear at each axis tick mark.

Programming Considerations

In most cases the order in which you use the Plotting System routines in your programs is unimportant. However, the following are exceptions:

- The Open Plotting System routine must be called before any other Plotting System routine.

- The Close Plotting System routine must be the last routine called.

- An output device must be opened, using the Assign Plotting System Output Device routine, before any output is attempted. Output occurs when the Output Currently Defined Chart routine, one of the immediate action graphic primitive routines (Arc, Arrow, Circle, Polyline, Polymarker, or Rectangle), or one of the Set Notation routines is called.

- An input device must be opened, using the Assign Plotting System Input Device routine, before graphic input is attempted. (Graphic input is accomplished by calling the Request Choice, Request Locator, and Request String routines. This is the only time an input device must be opened.) In addition, the same device that is being used for input must be opened as an output device before the input attempt if the input is to be echoed, as when using Request Locator or Request String. For example, by opening the display as both the input and output device and calling Request Locator, you can use the graphics cursor (+) to position the tip and end points of an arrow.

- When the same device is opened for both input and output, the Close Input/Output Device routine must be called twice before the Plotting System is closed.

- Any chart attributes must be set before the Output Currently Defined Chart routine is called.

- Any primitive attributes must be set before the primitive routine is called, since primitives and notation are immediate action routines. For example, Set Notation Color must be called before Define Notation String and Location if the color is to have an effect on the notation string.

- If primitives are to be used in conjunction with a chart, the Output Currently Defined Chart routine should be called before the primitive routine.

Example

To illustrate the capability of the Plotting System Library let's work through an example and build a chart. Since there are four different language bindings, our example will be written using a kind of "pseudocode" which shows the structure and logic without forcing us into a language-specific implementaion.

Suppose a department manager wants to create a chart of expenses by department so that it can be presented to each of four different departments: Engineering, Documentation, Administration and Marketing. He decides to plot the data as a bar chart with each department represented by a different bar. He also wants titles and axis tick labels so that each staff can recognize its related bar and cost figure.

The pseudocode example shows how the data values are defined for two arrays containing coordinates for four data points. XRAY provides X-axis coordinates and YRAY provides Y-axis coordinates. By default, the axis system of this chart is just large enough to accommodate the data.

The example also shows the routines used to create the bar chart. The Plotting System is opened and the plotter is assigned as the output device.

```
DATA XRAY /1.0, 2.0, 3.0, 4.0/
DATA YRAY /15.0, 27.0, 20.0, 11.0/
Open Plotting System
Assign Plotting System Output Device
("PLOTTER")
```

Next, we define the title of the chart, the dependent axis title, and the axis tick labels.

```
Define Title String ("EXPENSES")
Define Axis Title String (2,"Dollars in
Thousands")
Set Axis Labels (1,"/Eng/Doc/Adm/Mkt//")
```

The dollar sign labels for the independent axis are specified by entering:

- Two zeros for the parameters related to the independent axis
- "3" to specify dollar signs as the dependent axis tick labels
- "1" to indicate an integer value for the numeric data

The Plotting System ignores the zeros because character labels have already been set for the independent axis.

```
Set Axis Tick Label Type (0, 0, 3, 1)
```

The chart area frame and the view area frame are specified as visible and 0.1 percent of the horizontal view area width.

```
Set Existence of Chart Area Frame (1, 1)
Set Existence of View Area Frame (1, 1)
```

The data set is defined as a bar chart, the solid fill style is chosen, and the chart is sent to the plotter. Then the plotter and the Plotting System are closed.

```
Define X/Y Data Set (1, 4, XRAY, YRAY)
Define Data Set Chart Type to be Bar (1)
Set Output Primitive Style for Date Set (1, 2)
Output Currently Defined Chart
Close Input/Output Device ("PLOTTER")
Close Plotting System
```

When this program is translated into the appropriate language, compiled and linked with the appropriate Plotting System Library, and executed, it will produce the chart shown in the figure.

A number of default values were provided by the Plotting System in this example and they create a perfectly acceptable chart. But the programmer could have chosen a different fill area style and color for the bars, different colors for the axes, different color, size, or style for the title text, etc. And of course the chart could have been a line graph or pie chart (the Plotting System lets you explode a slice of pie to highlight it from the rest). Or if more than one data set were involved, you might want a stacked-bar chart.

The manuals supplied with the Plotting System Library are comprehensive and contain a section on how to select the type of chart which best fits your data, as well as all the details on each of the supplied functions. Also included with the package are a set of example charts written in each of the four languages to show you quickly how to start writing programs of your own. In fact we've just gone through one of them in pseudocode.

As in the other Professional Graphics Series packages which use the VDI, you can redirect the output of your program to other devices without changing any of your code. Simply use the DOS SET command to change the output device to a different driver.

Plotting System Functions

Following is a list of the functions built into the Plotting System Library.

- Chart type – determines basic format

    Text-only
    Bar
    Line
    Scatter
    Schedule
    Step
    Area
    Pie

- Chart attributes – determines control

    Line/fill style
    Line/bar width
    Bar/pie slice outline
    Text height
    Text font
    Legend
    Multiple charts per page
    Chart and view area

- Graphic input classes

  Request Locator
  Request String
  Request Choice

- Graphic output primitives

  Polyline
  Polymarker
  Rectangle
  Circle
  Arc
  Text
  Arrow

- Control

  Open Plotting System
  Close Plotting System
  Assign output device
  Assign input device
  Close input/output device
  Set display surface units
  Set display surface size
  Set view area extents
  Set foreground color
  Set background color
  Set horizontal or vertical chart
  Set baseline
  Set existence of axis
  Set existence of grid lines
  Set existence of view area frame
  Set existence of chart area frame
  Set axis type
  Set axis indent
  Set axis labels
  Set axis extents
  Set axis tick label type
  Reset entire plot environment to default
  Reset plotting environment
  Output the currently defined chart

- Data set definition

  Define sequenced data set
  Define X/Y data set
  Define schedule chart data set
  Undefine a data set

- Data set attribute definitions

  Name data set
  Set date
  Set visibility of a data set
  Define data set chart type to be bar
  Define data set chart type to be line
  Define data set chart type to be step
  Define data set chart type to be scatter
  Define data set chart type to be schedule
  Define data set chart type to be pie
  Set output primitive style for data set
  Set output primitive color for data set
  Set output primitive width for data set
  Set output primitive outline for data set
  Stack a set of data sets
  Fill between two data sets
  Set explode of pie slice

- Text definition

  Define title string
  Set title height
  Set title font
  Define sub-title string
  Set sub-title height
  Set sub-title font
  Define axis title string
  Set axis title height
  Set axis title font
  Set legend font
  Set existence and location of legend
  Set legend alignment
  Set notation string and location
  Set notation color
  Set notation font
  Set notation height
  Set notation alignment

- Inquiry

  Inquire on data set
  Inquire on device
  Inquire on Plotting System error
  Inquire on software system error

IBM Professional FORTRAN

Objectives

The primary objective in offering a new FORTRAN compiler for the Personal Computer was to provide a language capable of supporting large complex engineering and scientific applications, including programs already available for mainframe computers. Four other objectives were identified as being necessary to accomplish this goal: a FORTRAN language conforming to the full ANSI 77 standard; ease of migration from mainframe FORTRANs; high performance execution with acceptable compile time and high accuracy of numeric results; and ease of use.

The IBM Personal Computer Professional FORTRAN Compiler is a full function FORTRAN 77 compiler, designed according to the specifications of the ANSI Standard X3.9-1978 at the full level and providing significant compatibility with larger computer FORTRANs. Extensions to the standard ease conversion problems from FORTRAN 66 and provide heavily used facilities not included in the standard. Large programs may be migrated to the IBM Personal Computer with little or no change to the source code. Professional FORTRAN supports data seqments and individual arrays greater than 64KB, and program code can be as large as DOS will support, as long as no program unit is greater than 64KB.

Object code produced by Professional FORTRAN is optimized for execution time performance, using the same techniques common in mainframe compilers. The compiler was designed specifically for use with a math Co-processor, further assuring excellent performance and accuracy.

The compiler is easy to install and use. The convenience of the user was considered in the design of the compiler interface and the documentation. A comprehensive interactive debug package allows debugging at the source level, a feature usually reserved for large system compilers.

Prerequisites

- IBM PC, PC XT Portable PC with an IBM 8087 Math Co-processor or IBM Personal Computer AT with an IBM 80287 Math Co-processor for both compile and execution
- 192KB main storage for compile
- A fixed disk and one double-sided diskette drive or two double-sided diskette drives
- At least 64KB main storage and a Math Co-processor for execution
- IBM Personal Computer Disk Operating System Version 2.1 or later
- IBM Personal Computer Linker Version 2.3 or later

Using Debug increases the execution time storage requirement by approximately 90K bytes. The proper linker is shipped with the compiler.

Extensions to the ANSI Standard

Extensions to the ANSI FORTRAN 77 standard were provided for the following reasons: to facilitate conversion of FORTRAN 66 programs; to accommodate popular mainframe extensions to FORTRAN 77; and to improve usability of the compiler.

Two extensions aid in the conversion of FORTRAN 66 programs. A compiler switch causes DO-loops to be interpreted in FORTRAN 66 mode, with the loop test at the end of the loop. (However, no support is provided for extended DO loops.) Hollerith constants can be defined and assigned to numeric and logical data types. They can appear in DATA statements and as arguments in CALL statements, and they can be used in format specifications.

Several extensions match those provided by many mainframe FORTRAN compilers. Optional length specifications LOGICAL*1, LOGICAL*4, INTEGER*2, INTEGER*4, REAL*4, REAL*8, and COMPLEX*8 are provided for use in numeric and logical type specifications. Hexadecimal constants can be used to represent either numeric or character data. An INCLUDE statement allows secondary files to be merged with the source. Bit manipulation functions are implemented as defined in the proposed Industrial Real-Time FORTRAN Standard. Additional entry points are provided to maintain compatibility with the earlier ANSI/ISA bit function definitions.

Several extensions were implemented to improve compiler usability. Local symbols may contain up to 31 characters. Lower case characters are automatically promoted to upper case except when they appear within quotes or in a Hollerith specification. Speed of executable code may be improved and the size of the program reduced, where appropriate, by using a compiler option to treat all integer data as INTEGER*2. Conditionally compiled statements allow the programmer to include statements in his program which are either compiled or treated as comments based on a compiler switch. Library routines are provided to access the processor date and time.

Code Migration

Migration of programs from mainframe computers is made easier by the implementation of the full ANSI 77 standard and by including frequently used extensions. Extensions to the ANSI standard have already been discussed. The implementation of compiler options by switches rather than metacommands makes it easier to maintain programs which run on more than one computer.

Using the Professional FORTRAN Compiler, it is possible to compile and run very large programs on the IBM Personal Computer. The design of the 8088 family of processors requires main storage to be accessed in 64KB segments. Code and data are by default addressed by different segment registers. The restriction does not pose a real problem for code segments. While a single program unit cannot exceed 64KB, a program can consist of many program units accessed by far calls. If 50 bytes of generated code per source line are assumed (several times the likely amount), a program unit could be 1300 lines long without exceeding the 64KB restriction. That is much too long to conform to good programming practice.

The IBM Professional FORTRAN Compiler supports both data areas and individual arrays of greater than 64KB. Since a maximum of 64KB can be addressed using a single segment register, there must be a performance penalty for supporting large arrays. A compiler switch allows the user to generate code capable of accessing large arrays only for routines requiring it. The switch applies only to subroutines using adjustable or assumed-size arrays. In all other cases, the size of an array is known at compile time, and the compiler generates appropriate code.

For further information concerning code migration, please see "Portability and Conversion of FORTRAN Programs" in this document.

Performance and Accuracy

One of the top priorities in designing the IBM Professional FORTRAN Compiler was to produce efficient enough object code to make migration of code from mainframe computers attractive. The same optimization techniques used in mainframe compilers were used in the Professional FORTRAN Compiler: constant arithmetic and constant folding; common subexpression elimination; invariant code movement; strength reduction; and local optimization. Constant arithmetic and constant folding are operations performed at compile time on values which are constant at the point of reference. Common subexpression elimination calculates temporary values for subexpressions which are used in more than one expression. Invariant code movement extracts computations from DO-loops if they would be performed identically on every loop. Strength reduction is the substitution of one operation or set of operations for another in order to improve performance (for example, adding a number to itself instead of multiplying by 2 or using successive multiplies instead of the exponential function.)

Local optimization encompasses many techniques applying to both the main processor and the math Co-processor. Some are common to other compilers, but others are designed to take advantage of the specific hardware being used. The DO variable is stored only when necessary. If possible, code is structured so that operands to be accessed more than once are maintained in registers, a technique which especially applies to the register stack in the math Co-processor. The INC and DEC instructions and immediate operand forms of instructions are used where possible. Since main processor index registers are not interchangeable, the choice of registers and freeing of registers for particular types of operations is very important.

Use of the math Co-processor assures both fast and accurate floating point operations but creates a new set of challenges for optimization. For example, some floating point operations only apply to the top of the floating point stack, making the assignment of operands to registers more complex.

Results produced by the math Co-processor can be expected to be as accurate as on the System/370. The math Co-processor implements the IEEE floating point standard. Because of the way floating point numbers are stored in short and long floating point format, the math Co-processor can be expected to provide slightly more precise results in short format (single precision) and slightly less precise results in long format (double precision). However, a real but unpredictable increase in precision occurs because of the extended precision capability of the math Co-processor during intermediate calculations. The result probably will be overall precision at least equal to that of the System/370. The range of single precision numbers is smaller using IEEE floating point format, but the range of double precision numbers is much larger.

The routines in the intrinsic library make full use of the math co-processor, resulting in very fast execution.

Ease of Use

Three major areas were addressed in making IBM Professional FORTRAN easy to use: documentation; compiler facilities, including the Debug package; and compiler listings. The decision to provide a Library Manager as part of the compiler package is an additional aid to the user, since it enables him to build libraries of common subroutines.

Two manuals are provided with the compiler:

Professional FORTRAN Installation and Use

and

Professional FORTRAN Reference.

There is a color-coded "Quick Start" chapter which enables a new user to install the compiler and begin compiling programs immediately. Although neither manual is meant to be a tutorial, every effort has been made to present material clearly and in a logical manner. Extensions are well documented, and there is an appendix providing helpful information on converting existing programs and writing new programs in such a way that they will be portable. Clear explanations are provided for error messages, including examples of possible causes.

The compiler provides several facilities which aid the user in debugging his program. Conditionally compiled statements may be either compiled or treated as comments depending upon a compiler option. Default file assignments allow program checkout without providing actual file names. The Interactive Symbolic Debug Program allows program debugging at the symbolic level while the program is actually running.

Any program or subprogram may be compiled with the debug option. If a subprogram is to be debugged interactively, all program units in its chain of calls must also be compiled using the debug option. Using the debug option suppresses code optimization, inserts calls to debug routines, and causes a symbol table to be included in the object module. Once a program unit has been compiled with the debug option, the user may run the program under control of the Debug program, which allows him to: set breakpoints to monitor program flow or check values of variables; step through the program unit statement by statement; examine or change the values of variables; trace the flow of individual statements, a range of statements, or entries to and exits from subprograms; log the results of a debug session.
Printed output from the compiler also contributes to rapid program checkout. Diagnostics are inline and undermarked. Even if the "errors only" compiler option is chosen, the line in error is printed before the error message or messages. All lines are numbered, so that they correspond to the line numbers of the original source during editing. Error messages are descriptive, and more complete explanations are provided in the manual. A cross reference can be requested, containing all symbols and line numbers and indicating the type of use for symbols (definition, reference, or value change.)

Conclusion

The original design criteria for the compiler – a full ANSI 77 FORTRAN implementation, extensions which contribute to portability of mainframe code, execution speed and accuracy, and ease of use – have been met. The resulting product provides a suitable vehicle for the design and checkout of a wide range of sophisticated engineering and scientific FORTRAN applications.

Portability and Conversion

If you have an existing library of FORTRAN programs, or if the programs you write must run on more than one computer, you will have some additional concerns about using IBM Professional FORTRAN. The following issues affect the portability of programs:

- Differences in language level
- Differences in hardware (especially word size)
- Use of language extensions
- Implementation differences.

Language Level

Theoretically, most portability problems can be eliminated by following the ANSI standard. However, three standards are in common use: FORTRAN 66, which is specified by the document ANSI X3.9-1966; AND FORTRAN 77 and Subset FORTRAN 77, both specified by the document ANSI X3.9-1978.

Many existing programs were written in variations of FORTRAN 66, and these present the biggest challenge for conversion to IBM Professional FORTRAN. Some features which were part of FORTRAN 66 are no longer supported (DEFINE FILE, for example). Some common extensions to FORTRAN 66 are supported by FORTRAN 77, but not necessarily with the same syntax. At least one major statement, the DO, looks the same but is implemented differently. The following paragraphs address several of the commonly encountered problems.

DO-Loops

In FORTRAN 66, any DO-loop was executed at least once, since the implementation put the test at the end of the loop. In FORTRAN 77, the test is at the front of the loop, and the loop might not be executed at all. FORTRAN 66 also allowed extended DO-loops; the program could exit from the innermost loop and return to the same point. Using the /F compiler option causes all DO-loops to be evaluated according to the FORTRAN 66 standard. No support is provided for extended DO-loops. You must restructure any program using that technique, perhaps by using an equivalent IF statement.

EQUIVALENCE

The FORTRAN 66 standard permitted arrays having more than one dimension to appear in EQUIVALENCE statements with only a single subscript. This is no longer permitted, and no conversion aid is provided.

Hollerith Data

In FORTRAN 66, Hollerith data could be assigned to integer, logical, real, or complex data types. A character type is provided in FORTRAN 77, and the assignment of character data to other variable types is no longer permitted by the current standard. IBM Professional FORTRAN provides Hollerith constants as an extension. Hollerith constants can appear in DATA statements and as arguments in a CALL statement.

DEFINE FILE

The functions performed by the DEFINE FILE statement in FORTRAN 66 are provided by the OPEN statement in FORTRAN 77.

Subset FORTRAN 77

IBM Professional FORTRAN is an implementation of the full standard ANSI X3.9-178 with extensions. Any program conforming to the subset level as defined by the same standard, and not using any extensions, should compile properly using the IBM Professional FORTRAN compiler. However, it might not run properly because of differences in hardware and implementation.

Hardware

Differences in hardware dictate differences in implementation. They are considered here as a separate topic because the resulting implementation is more a matter of necessity than choice. Hardware differences can be internal or external. Internal considerations are word size, the number of characters in a word, and floating-point arithmetic. Since differences in the type and size of external storage devices are only one of the factors influencing I/O implementation, it will be discussed under "Implementation Differences."

Word Size

The ANSI FORTRAN standards define the length of the supported data types in terms of numeric or character "storage units" of unspecified size. A real, integer, or logical data item, for example, must occupy one numeric storage unit, and a double-precision data item must occupy two consecutive storage units. Considering only integer data, it is obvious that a larger number can be stored in a 36-bit word as opposed to a 32-bit word, although both are acceptable implementations of the integer type. If you are moving a program to the IBM Personal Computer from a computer having a word size larger than 32 bits, you must be sure that integer values are between –2**31 and 2**31-1.

Character Data in Other Data Types

A problem arises when numeric or logical data types are used to store Hollerith data, even though a language extension is provided to allow the use of that convention. Computers differ in the number of characters that can be stored in a word, because of both the size of the word and the number of bits required to represent a character.

The ANSI standard does not specify a correspondence between numeric and characters storage units. A program that uses a DATA statement to store ten 6-bit Hollerith characters in each 60-bit word will not compile correctly on the IBM Personal Computer, where four 8-bit characters are stored in a 32-bit word. In some cases, a program can be converted by using a larger size variable to store Hollerith data; REAL*8 instead of REAL*4, for example.

In the IBM Personal Computer, numbers are stored internally with the bytes swapped. For this reason, characters cannot be stored in numeric data types using a hexadecimal representation and later retrieved as characters. The problem can be resolved in the general case only by using the character data type. You should choose this method for new programs.

Floating-Point Arithmetic

Not only is the actual length of floating-point numbers not defined by the ANSI FORTRAN standards, but neither are the number base, exponent range, or number of significant digits. The range is controlled by the number of bits used for the exponent and the chosen base. The IBM 370, for example, uses a 7-bit biased exponent and a base of 16. The exponent always shows 63 (the bias) more than the actual value in order to provide a representation for negative numbers without using a sign. Therefore, the largest positive exponent that can be represented is 63 to base 16, or approximately 76 to base 10 for both single-precision and double-precision values.

The IBM Personal Computer with a math Co-processor implements the IEEE standard for floating-point arithmetic, which defines the exponent length as 8 bits biased by 127 for short real (single-precision) and 11 bits biased by 1023 for long real (double-precision) values. The largest possible positive exponent is approximately 38 to base 10 (127 to base 2) for single precision and 308 to base 10 (1023 to base 2) for double precision.

A program using single-precision arithmetic might produce legitimate results on the 370 that are out of range on the IBM Personal Computer. Using double precision might cause just the opposite result. There is no solution to this problem except to limit programs that must be run on more than one computer to those for which the expected floating-point range is valid on all the computers.

The number of significant digits (or precision) is controlled by the number of bits used for the fraction and by the base. The influence of the base is not obvious. It has to do with the fact that there are gaps between the numbers that can be represented exactly within the computer. The larger the base, the larger the gaps. If the number of significant bits is the same, the greatest precision is obtained by using a base of two.

You should be aware that a complex program can produce different results on one computer than another because of differences in internal precision. The difference can be minimized, but not necessarily eliminated, by using double-precision arithmetic and by using numerical methods that tend to retain maximum precision.

Some programs require the base or precision of the machine on which they are run as input. This applies to some random number generators and to programs that use iterative methods to arrive at a result, stopping when no significant change would result from further iterations. If this is the case, be sure that the constants are flagged in a way that they can be easily found and changed.

Language Extensions

If you are converting an existing program, you will find the process more difficult if extensions to the ANSI standard were used. If you are writing a new program, weigh the utility of extensions to the standard against any future portability problems. There have been many extensions to the FORTRAN language. Some extensions to FORTRAN 66 are standard in FORTRAN 77. Some are now implemented, but in a different manner. Only a few topics can be covered here.

Additional Data Types

Many compilers, including the IBM Professional FORTRAN compiler, provide extensions to the standard data types. Examples would be REAL*8 and LOGICAL*1. Some of these extensions have standard equivalents. DOUBLE PRECISION can be substituted for REAL*8, REAL for REAL*4, LOGICAL for LOGICAL*4, and INTEGER for INTEGER*4, in most cases.

Data types that are not standard are normally used to save time, space, or both. Unless they are made equivalent (in an EQUIVALENCE statement) to variables of a different type, they can usually be converted to the nearest standard data type. BYTE and INTEGER*2 become INTEGER, and LOGICAL*1 and LOGICAL*2 become LOGICAL.

Hexadecimal and Octal Data

Most FORTRAN compilers allow variables to be initialized with either hexadecimal or octal data. Some support both. The syntax used differs widely. For example, the same hexadecimal data can be entered as Z4FD1, Z'4FD1', or '4FD1'X, depending on the compiler. The compiler may supply zeros on the left or blanks on the right as padding, depending on the variable type.

IBM Professional FORTRAN supports hexadecimal data only. If you are converting an existing program or if your program must run on other computers, you might have to adjust to differences in syntax or implementation. A possible method is to supply comment cards to be transformed by a precompiler or text editor. For example:

```
CIBMPODATA VAR /Z'4FD1'/
CIBMVSDATA VAR /Z4FD1/
```

generates the following:

```
      DATA VAR /Z'4FD1'/
CIBMVSDATA VAR /Z4FD1/
```

for use on the IBM Personal Computer when a text editor is used to substitute blanks for all occurrences of the characters 'CIBMPC.'

Data Management Extensions

The FORTRAN language has been extended in almost all compilers to provide additional I/O facilities. Some of these extensions are:

- Asynchronous I/O supported with BUFIN/ BUFOUT, or with READ/WRITE ID=
- Internal files, supported by many vendors with ENCODE/DECODE
- File types other than sequential and direct, such as IBM's partitioned and VSAM files, and another manufacturer's keyed files
- Read and write keywords that address a specific kind of I/O equipment, such as ACCEPT, DISPLAY, and TYPE.

If you are converting programs that use these or other extensions, you must become familiar with the I/O facilities provided by each compiler.

Some functions supported by extensions to FORTRAN 66 are standard in FORTRAN 77. Two examples are list-directed I/O and internal files. The support may be similar or identical for list-directed I/O. Internal file transfers formerly performed by DECODE and ENCODE are performed by READ and WRITE statements in FORTRAN 77, and you must change these statements manually or use a preprocessor.

Converting a program that uses file types other than sequential and direct requires a careful analysis of the problem. Direct conversion may not be possible. IBM Professional FORTRAN does not support asynchronous I/O. If asynchronous I/O is not necessary, you can use ordinary synchronous I/O statements. Statements using ACCEPT, DISPLAY, TYPE, and other similar keywords can be replaced by appropriate READ or WRITE statements.

Data Initialization in Type Statements

Data initialization in type statements is not supported by IBM Professional FORTRAN and must be moved to DATA statements.

Compiler Metacommands

If the program you are converting contains any compiler directives, you must either remove them or modify them to conform to the requirements of the new compiler. The only compiler directive supported by IBM Professional FORTRAN is the INCLUDE command. If your program uses an INCLUDE statement, you must be sure that the syntax is correct and that only a single level is used. Other compiler directives are supported by compiler options. For example, the function of the $DO66 metacommand in IBM Personal Computer FORTRAN V2.00 is performed by the /F option in IBM Professional FORTRAN.

/I Compiler Option

In IBM Professional FORTRAN, characteristics of integer data can be controlled by the /I compiler option. The use of the /I option can significantly increase program speed while maintaining portability. It can cause portability problems if integer data is made equivalent to non-integer data, or if integer data is passed to subroutines. There are three types of integer data:

- INTEGER*2
- INTEGER
- INTEGER*4.

These will be abbreviated as I*2, I, and I*4 respectively.

Two new intrinsic functions have been added: HFIX and JFIX. They operate like INT or IFIX, but always return I*2 or I*4 type respectively. INT and IFIX are actually mapped into HFIX or JFIX, depending on the /I option.

Intrinsic functions with non-integer arguments that return an integer type, return the type of INTEGER. That is, the type varies with the setting of the /I option.

Intrinsic functions with a single integer argument, return the same type as their input. There is only one such function: ABS or IABS.

Intrinsic functions with multiple integer arguments, have all their arguments converted to the type of the largest argument. This is the result type as well, except for AMAX0 and AMIN0, which return type REAL.

Integer constants in the range:

```
±2**15 (-32,768) to 2**15±1 (32,767)
```

are typed I*2. Integer constants outside that range, but within:

```
±2**31 (±2,147,483,648) to 2**31-1 (2,147,483,647)
```

are typed I*4. Integer constants outside that range are in error. If an I*4 constant is scanned when the /I option is on, a warning message is produced, but the constant is still typed I*4.

Integer constants passed as arguments are converted to I*4 if the /I option is off. Otherwise, they are passed as described above. The HFIX or JFIX functions can override the integer constants. However, the use of HFIX and JFIX will cause portability problems.

Implementation Differences

Many differences among FORTRAN compilers arise because ANSI standards do not define such things as:

- Range or precision of numbers
- Character coding and collating sequence
- Physical properties of records and files
- Limits imposed by the implementation.

The range and precision of numbers is discussed earlier under "Hardware." The other items are discussed in this section.

Character Coding

The ANSI standard for FORTRAN 77 only specifies a full collating sequence for the lexical intrinsic functions LGE, LGT, LLE, AND LLT, which must use the collating sequence described in ANSI X3.4-1977 (the American National Standard Code for Information Interchange, or ASCII). Otherwise, the only requirement is that the letters A-Z and the numbers 0-9 are ordered in the normal manner without intermixing, and that the blank character is less than both A and zero.

The IBM Personal Computer uses an 8 bit ASCII character representation and the ASCII collating sequence. Many other computers use 6 bit ASCII or 8 bit Extended Binary Coded Decimal Interchange Code (EBCDIC). If you are converting programs that compare character data using the relational operators .LE, .GT., and so on, you must be sure that the code logic is preserved. Also, you might need to convert character data created by other programs to ASCII before you process it on the IBM Personal Computer. This can be done by a simple table lookup.

Records and Files

The physical storage of records and files is determined by the available storage media, the operating system, and the implementer. Often it is not possible to use a file created on one computer as input to a program on another, or even on the same computer if more than one operating system or compiler is involved.

Some problems are caused by incompatible media. Examples of diskette differences are:

- Physical size (8-inch or 5-inch)
- Same physical size but formatted differently
- Data density.

The only solution to this type of problem is to transmit the data directly from one computer to the other. Differences in word size made the exchange of unformatted data impractical. Generally, files that will be used on other computers must be formatted, and conversion from EBCDIC to ASCII (or vice-versa) might be required. Different record formats can be used by different compilers, especially on microcomputers.

> **Note:** The /R compiler option must be used at runtime if records greater than 1024K bytes are to be read or written. This restriction is imposed by the record transfer implementation.

Limits and Restrictions

The ANSI standards do not define “correct” implementation for such items as the number of continuation cards allowed by the compiler, the number of nested DO-loops supported, or the number of labels the compiler can handle without overflowing its internal table. See Appendix D, “Limits and Ranges,” in the *Professional FORTRAN Reference* manual for more information about the decisions that are left to the implementer. Most programs will not violate any of the implementation limits.

SAVE Statement

The SAVE statement is not implemented by IBM Professional FORTRAN, since all storage is treated as static. However, if you want your programs to be portable, you should use the SAVE statement (preferably without a list) as if it were fully implemented.

Environmental Differences

The FORTRAN implementation is affected by the underlying computer hardware, the operating system, and the linker. The following two examples illustrate the types of considerations you might have:

1. Some systems clear storage before loading a program. Some, such as DOS, do not. You must initialize every variable before using it.

2. In the IBM Professional Computer, only 64K can be addressed at one time using a single segment register. This imposes restrictions on how a program can be structured. For example, no single module containing more than 64K or code can be compiled on the IBM Professional FORTRAN compiler. Good programming practice suggests that each module should perform a single well-defined function. Thus, you should not be affected by this restriction when writing new code, but you might have to restructure a program you are converting. If you have subroutines with adjustable or assumed-size arrays, you must compiler them using the /B compiler option, if they will ever be passed arrays greater than 64K bytes in length. This will process all adjustable and assumed-size arrays to be larger than 64K bytes.

Suggestions When Writing or Converting Programs

If you are writing new programs and are concerned about portability, consider the following suggestions:

1. Use only features that conform to the ANSI FORTRAN 77 standard.

2. If you are using more than one type of microcomputer, determine whether you need to restrict your code to the FORTRAN 77 subset standard.

3. If you use any language extensions, mark them clearly.

4. Flag any machine-dependent constants or parameters such as base or precision.

5. Use standard naming conventions (integer names begin with I-N, and so on) so that default sizes can be changed easily by the IMPLICIT statement.

6. When possible, use the /I compiler option instead of defining variables as INTEGER*2.

7. Initialize all variables before use.

8. Avoid hardware-dependent code.

If you are converting existing programs, use the following guidelines:

1. Become familiar with both compilers and systems before attempting conversion.

2. Consider writing a precompiler if you have many programs to convert.

3. Make use of the global change facility in a good text editor.

4. If time allows, make any changes that will simplify conversion the next time, following the suggestions for writing new programs.

IBM PC Data Acquisition and Control Adapter and Software

Introduction

Personal computers have become common tools in business and industry over the past few years. These computers have begun to be integrated into workstations to allow more creative and productive use of them in the laboratories of scientists and engineers. In order to aid these users, IBM has developed hardware and software products resulting in the IBM PC Engineering/Scientific Series. These products allow the user to customize a PC into a personal, versatile, productive tool that can be used in both the laboratory and the office.

Two Hardware Products

Two products allow direct connection of IBM Personal Computers to instrumentation, sensors and control signals.

The IBM PC Data Acquisition and Control (DAC) Adapter provides the capability to control and monitor processes within a sensor based system. Also available as an option for use with the DAC Adapter is a DAC Adapter Distribution Panel.

The General Purpose Interface Bus (GPIB) Adapter which allows access to and control of instruments and devices that are compatible with the IEEE-488 standard.

Two Software Products

A programming support product is also provided for the DAC and GPIB Adapters to enable efficient application development.

The IBM PC Data Acquisition and Control Adapter Programming Support provides an easy to use interface for accessing the analog and digital I/O onboard and expansion capabilities of the DAC Adapter.

The IBM PC General Purpose Interface Bus Adapter Programming Support provides appropriate levels of capability for both primitive functions that handle GPIB activities and high level functions that contribute to ease of use. The result can be a highly efficient workstation that can help improve accuracy and productivity in even complex testing or measurement systems using the GPIB Adapter.

The IBM PC DAC and GPIB Adapters and their programming support products enable engineers, scientists and technicians to use personal computers from IBM in both process and instrument control. The capabilities of the adapters and the real time software coupled with the open architecture and versatility of the line of IBM Personal Computers, offer new opportunities for designing workstations that can support data acquisition, control, analysis and quality control testing activities in the lab, the pilot plant or the full-scale production line. In addition to the ease of use of obtaining data and automating the laboratory through the IBM PC DAC and GPIB Adapters, the user can utilize the power of the IBM PC family for data analysis and data base management.

The remainder of this article will be dedicated to a description of the IBM PC Data Acquisition and Control Adapter and its software suuport. The next article will describe more fully the IBM PC General Purpose Interface Bus Adapter and its software support.

IBM Data Acquisition and Control Adapter

The engineering/scientific workstation can use the IBM PC DAC Adapter to interface the digital computer with analog laboratory testing and process control equipment. Using a single adapter and appropriate software enables the IBM PC to monitor and control analog and discrete signals. Four DAC Adapters can be used with a single IBM Personal Computer.

Data is acquired and processes controlled by the DAC Adapter which integrates analog, binary and time/counter devices on a single adapter card, which plugs into any full length slot in an IBM PC AT, IBM PC XT, IBM PC or IBM Portable PC. Generally, the information being read by the computer is analog in nature. In order for the computer to be able to process this information it must first be translated into digital data. This is the function of the analog-to-digital (A/D) converter. The digital-to-analog (D/A) converter and binary I/O devices allow the user to control the operation of any process.

Analog-To-Digital and Digital-To-Analog Conversion

An analog-to-digital conversion subsystem or analog input will convert voltages over a given range to digital values. The three selectable ranges allowed are:

−5 to +5 volts / −10 to +10 volts / 0 to +10 volts

The digital-to-analog subsystem or analog out converts digital values to the selectable values referenced above.

Binary Input/Output

A binary I/O device consists of two subsystems, one for input and one for output. The device has a 16-bit input port, a 16-bit output port, I/O handshaking capabilities and optional external clocking/ triggering. This device will sense the state of discrete signals and control devices that require control signals.

Timer/Counter Device

The 8253-5 timer/counter device which can be programmed includes three 16-bit timer/counters: 0, 1 and 2. These timer/counters are used to provide pulses, count events when pulsed and, through software, generate interrupts to the DAC Distribution Panel. Counters 0 and 1 are cascaded together to perform internal clocking functions. Counter 2 is used as a countdown timer.

Additional Features

Additional user features of the DAC Adapter include:

- A sytem reference voltage.
- Optional screw terminal DAC Distribution Panel for easy access to I/O signals.
- Diagnostic wrap connector for easy installation verification and ongoing reverification of I/O signal levels.
- Up to four IBM DAC Adapters in a system.

The DAC Adapter features an exspansion bus interface that consists of two 34-pin transition connectors and provides a full 16-bit parallel data path and addressing for up to 253 exspansion devices.

IBM PC DAC Adapter Interface Specifications

The Data Acqusition and Control Adapter utilizes IBM PC I/O channels as described in the Technical Reference Manuals PN 6025005 and PN 6936808.

Analog Input Characteristics

- Resolution is 12 bits
- Input channels: 4 differential
- Input ranges: Switch Selectable 0 to +10 volts (unipolar), −5 to +5 volts (bipolar), −10 to +10 volts (bipolar)
- Digital coding: unipolar: binary: offset binary
- Safe input voltage: +/− 30 volts minimum input voltage (power on or off)
- Input current: +/− 4 milliamps at maximum input voltage
- Common mode input range: +/− 11 volts maximum
- Common mode rejection: 72 db minimum ratio (signal within common mode range)
- Differential linearity error: +/− LSB maximum
- Differential linearity stability: +/− 5ppm/degree C maximum, guaranteed montonic
- Gain error: +/− 0.1% maximum between ranges: any range adjustable to zero
- Gain stability: +/− 32 ppm/degree C of FSR maximum
- Unipolar offset error: adjustable to zero
- Unipolar offset stability: +/− 24 ppm/degree C of FSR maximum
- Bipolar offset error: adjustable to zero
- Bipolar offset stability: +/− 24 ppm/degree C of FSR maximum
- Input resistance: 100 megohms minimum
- Input capacitance: 200 picofarads maximum measured at the distribution panel connector

- Input leakage current: +/− 300 nanoamps maximum
- Settling time: channel acquisition: 20 microseconds maximum to within +/− 0.1% of the input value
- Conversion time: 35 microseconds maximum
- Throughput to memory: 15,000 conversion/second minimum
- A/D CE input impedance: one LS TTL load plus 10 kilohm pull-up resistor
- A/D CO fanout: 10 LS TTL loads or 2 standard TTL loads

Analog Output Characteristics

- Resolution: 12 bits
- Output channels: 2
- Output ranges: switch selectable: 0 to +10 volts (unipolar) −5 to +5 volts (bipolar) −10 to +10 volts (bipolar)
- Digital coding:unipolar:binary;bipolar:offset binary
- Output load current: +/− 5 milliamps minimum
- Output load capacitance: 0.5 microfarads maximum for stability
- Output protection: protected for short to common
- Output impedance: 2 ohms maximum at distribution panel connector
- Differential linearity error: +/− 1/2 LSB maximum, guaranteed monotonic
- Gain error: +/− 0.1% maximum between ranges; any range adjustable to zero
- Gain stability: +/− 35 ppm/degree C of FSR maximum
- Unipolar offset error: +/− 3.25 millivolts maximum
- Unipolar offset stability: +/− 8 ppm/degree C of FSR maximum
- Bipolar offset error: adjustable to zero
- Bipolar offset stability: +/− 24 ppm/degree C of FSR maximum
- Power supply rejection: +/− 1/2 LSB maximum change in full-scale calibration
- Throughput from memory: 25,000 conversions/second maximum
- Dynamic characteristics for a +/− 10 volt step with less than +/− 5 milliamps and less than 1000 picofarad load
- Overshoot: +/− 1% of FSR maximum
- Settling time: 10 microseconds maximum to within +/−0.1 % FSR

Binary Input BI0 Through BI15

- Input impedance: one LS TTL load plus 10 kilohm pull-up resistor
- Throughput to memory: 25,000 operations/second minimum

BI Hold

- Input impedance: two LS TTL loads plus one 10 kilohm pull-up resistor

BI STROBE

- Input impedance: one LS TTL load plus one 10 kilohm pull-up resistor

BI CTS

- Fanout 10 LS TTL loads or 2 standard TTL loads

Binary Output BO0 Through BO15

- Fanout: 28 LS TTL loads or 7 standard TTL loads
- Throughput from memory: 25,000 operations/second minimum

BO GATE

- Input impedance: two LS TTL loads plus one 10 kilohm pull-up resistor

BO CTS

- Input impedance: one LS TTL load plus 10 kilohm pull-up resistor

BO STROBE

- Fanout: 10 LS TTL loads or 2 standard TTL loads

32-Bit Timer Device (Counters 0 and 1)

COUNTER 0

- CLK 0 Frequency: 1.023 MHz

RATE OUT

- Fanout: 10 LS TTL loads or 2 standard loads

COUNTER 1 DELAY OUT

- Fanout: 10 LS TTL loads or 2 standard TTL loads

16-Bit Counter Device

COUNT IN

- Input impedance: one LS TTL load plus 10 kilohm pull-up resistor
- Input frequency: DC −2 MHz (50% duty cycle)

COUNT OUT

- Fanout: 10 LS TTL loads or 2 standard TTL loads

IBM PC DAC Adapter Distribution Panel

- Optional accessory for easy access to I/O signals, voltages and ground of the adapter
- Printed circuit board contains 4 barrier-type screw terminal strips for a total of 88 terminations
- Permanently attached shielded flat cable with 60-pin connector
- Cable and board assembly are housed in a metal enclosure with a slot for easy entry and exit of user cabling
- Meets FCC Class B requirements

Diagnostics

A diagnostic program to test the functionality and installation of the hardware is provided with each IBM PC DAC Adapter. The diagnostic requires a wrap connector which is provided with the adapter.

Power Requirements

- +5 volts +/− 5% at approximately 1 amp typical (1.5 amps maximum)

System Reference Voltage

- Output voltage: +10 volts
- Accuracy : +/− 1.2%
- Output load current: +/− 2 milliamps maximum
- Output load capacitance: 0.5 microfarads maximum at stability
- Output protection: protected for short to common
- Output impedance: 2 ohms maximum at distribution panel connector

Dimensions

- Length: 335.3mm (13.2in.)
- Height: 99.1mm (3.9in.)

Operating Environment

- Operating temperature: + 15.6 to 32.2 C (+60 to +90 F)
- Relative humidity: 8 to 80 % (non-condensing)
- Altitude: 2187m (7000 ft.) maximum

Setting Interrupts

Interrupts can be set via dips switches on the DAC Adapter. You have the ability to select interrupt request levels IRQ3 through IRQ7.

Requirements

The IBM PC DAC Adapter may be used with an IBM Personal Computer AT, the IBM PC XT,the IBM PC or the IBM Portable PC. The IBM DAC Adapter Programming Support requires the IBM Disk Operating System (DOS) 2.00 or higher and the IBM PC DAC Adapter. The IBM Personal Computer AT requires DOS 3.00.

IBM Data Acquisition and Control Adapter Programming Support

The IBM PC DAC Adapter Programming Support provides an easy-to-use interface for accessing the onboard and expansion analog and digital I/O capabilities of the adapter. The support operates under the IBM Disk Operating System (DOS) 2.00 or higher, and provides a device driver enabling the user of IBM PCs to configure the workstation. For development, there are sample programs in Interpretive BASICA, Compiled BASIC, Lattice C, FORTRAN 2.0 and IBM Professional FORTRAN.

Included are many useful programming support features through the use of function calls. Entire arrays of data that are input or output can be accomplished by one function call. Such functions are provided for analog input/output, binary input/output and for counter input. There are scanning functions that can scan a range of analog input channels, sampling functions that can be specified in samples per second, and full support of the expansion capabilities of the adapter. Examples in each of the supported languages for each of the functions are available in the DAC Adapter Programming Support manual.

Functions

When programming, three types of functions can be used:

- Input Functions collect input data and move it to memory
- Output Functions move data from memory to an external device
- Utility Functions control counters/timers and program execution

The software performs these functions according to rate or frequency. These functions may be further classified into:

- SIMPLE FUNCTIONS. Also called non-interactive functions. These execute only once.
- MULTIPLE FUNCTIONS. These iterative functions execute according to the number of times specified by a count argument in the function. A rate argument governs the rate of execution.
- SCANNING FUNCTIONS. Scans are sets of single inputs collected from a range of consecutively numbered channels. These samples are taken as close together as the device allows.

Function Names

Function names reflect the function type and the device involved. The first letter signifies the type of device:

- A analog device
- B binary device
- BIT a binary function that works with only a single bit of a binary word
- C counter device

The next two letters indicate the input output:

- IN Input function
- OU Output function

The last letters show the rate types:

- S simple function
- M multiple function
- SC scanning instruction

Analog - Binary - Counters - Delay Functions

The analog I/O functions are:

- AINM Analog Input Multiple. Inputs values from the adapter, device and channel at the specified rate.
- AINS Analog Input Simple. Inputs a single value from the adapter, device and channel.
- AINSC Analog Input Scan. Input values from multiple channels on the input device at the rate specified.
- AOUM Analog Output Mulitple. Outputs a range of variables to the adapter, device and channel at the rate specified.
- AOUS Analog Output Simple. Outputs a single variable to the adapter, device and channel.

The Binary I/O Functions are:

- BINM Binary Multiple. Inputs selected states of the binary input word at the rate specified.
- BINS Binary Input Simple. Inputs a single binary word from the adapter and device.
- BITINS Binary Input Bit Simple. Inputs the state of a bit in the binary input word.
- BITOUS Binary Bit Output Simple. Outputs the state of a bit in the binary output word.
- BOUM Binary Output Multiple. Outputs selected binary state variables at the rate specified.
- BOUS Binary Output Simple. Outputs the contents of the data variable as a binary word.

The Counter Functions are:

- CINM Counter Input Multiple. Reads counter values from the adapter at the rate specified.
- CINS Counter Input Simple. Reads the adapter counter value.
- CSET Counter Set. Initializes the adapter counter.

The Delay Functions is:

- DELAY Delay execution. Delay application program execution.

Performance

The user can extend device driver performance through the use of the extended execution mode operation. The following data rates can be achieved:

AINM	15 K Samples/ Second
AOUM, BINM, BOUM	17 K Samples/ Second
CINM	12 K Samples/ Second

Configuration Requirements

The minimum configuration allowed includes:

Machine Requirements

- An IBM PC with a minimum of 64 KB of memory (BASIC, C, FORTRAN compilers require additonal)
- At least one 320 KB diskette drive
- An IBM monitor
- The DAC Adpater
- Any cables necessary to connect the adapter to the device it is to interface.

Programming Requirements

- IBM PC Disk Operating System, Version 2.00 or later
- IBM PC DAC Adapter Programming Support
- Any of the following optional languages:

  IBM PC BASICA Interpreter

  IBM PC BASIC Compiler

  IBM FORTRAN Compiler Version 2.00

  IBM PC Professional FORTRAN Compiler

  C Lattice Compiler Version 2.0

Adapter Dependent Information

The adapter switches are set to give the computer and software the following values:

- Interrupt level
- Adapter number
- Analog input and output ranges.

Setting interrupts

Unlike some Personal Computer peripherals, the DAC Adapter and Programming Support uses a shared interrupt system. While you can use vectored and shared interrupt devices in the same system, no vectored interrupt can have the same priority as that of the adapter. The adapter can use interrupt request levels IRQ3 through IRQ7.

Assigning adapter numbers

Each adapter has a number that identifies it to the system. This is the adapter number.This number is included in the argument list of any function call that uses the adapter. Before installing the adapter, set the switches to assign the number (0-3).

Setting analog ranges

Adapter switches govern the voltage range of the analog input and analog output devices. To accurately compute analog I/O values, you must know the full-scale range of your analog I/O hardware. These ranges are:

−5 to 5 volts

−10 to 10 volts

−0 to 10 volts

BASIC Sample Program

The following paragraphs contain the sample program description and the Interpretive BASICA code to demonstrate the programming of selected I/O function calls supported by the DAC Adapter Programming Support software. Each section of the BASIC sample program illustrates the use of one analog I/O function. The program begins with a header and concludes with brief error-handling and data-plotting routines. The header executes first.The program pauses, displaying a prompt until you press the Enter key. Then Section 1 runs. Sections 2 through 5 work in the same way.

Requirements

The BASIC sample program requires the following:

- Analog input and analog output ranges set to −5 to +5 volts
- I/O address for the adapter set to 0
- Color monitor for Sections 4 and 5
- Test circuit
- Distribution Panel.

Wiring the Test Circuit

The sample program requires a test circuit. You use this to run the sample program without connections to external devices. The circuit provides analog input signals which it derives from internal voltages on the adapter. It also allows you to see, by varying the intensity of an LED, the effect of changing analog output voltage.

The test circuit components required are:

- A 100 K ohm potentiometer.
- A standard Light Emitting Diode (LED) with a 10 K ohm resistor ballast.
- Three clip leads.
- 24 inches of 17 gauge insulated, stranded wire. Use this wire to make six 4-inch jumpers, stripped 1/2-inch at each end.

Wire the test circuit to the DAC Adapter Distribution Panel as specified in the following table.

From	To
D/A 0	Pot end
D/A0	LED + (red)
D/A 1	Pot end
AGND	LED - (black)
AGND	A/D 0-
AGND	A/D 1−
A/D0	BO8
A/D1	Pot wiper

Adding the Header

The BASICA program requires a header supplied on the DAC Adapter Programming Support Distribution Diskette. This header is a pre-written section of BASICA code. It must appear at the beginning of each program. The header code is not included in the sample program listing.

After the header executes, every DAC Adapter software function is accessible as a real variable.

Global Variables

The sample program sections use global variables of their own. You must initialize them as shown here:

```
120 ADAPT% = 0
140 'use on-board analog I/O device
150 DEVICE% = 9
160 'and DIMension a 640-point array
170 'to be used in sections 3,4, and 5
180 DIM RAWDATA% (639)
```

For the purposes of these samples, assign address 0 to the adapter.

All of the sample program sections deal with the on-board analog I/O device. Because of this, the programs set DEVICE%=9. They also DIMension an integer array that will hold 640 data points. The number 640 is somewhat arbitrary. The simple data plotting routine used by several of the samples has a 640-point horizontal resolution.In actual practice, you will probably need to dimension all types of arrays at various points in the program. To provide a clear sample, these programs use a single array, named RAWDATA%. This array stores all data read and written by multiple analog I/O functions (AINM, AINSC, and AOUM).

Section 1: A Simple Analog Output from Two Channels

This section uses the function AOUS (Analog Output Simple) to:

- output voltages of +5 Volts to analog output channel 1
- output −5 Volts to analog output channel 0.

These voltages serve as the source of the analog signals that are used in the subsequent sections.

The first step is to assign values for the remaining arguments of AOUS (ADAPT% and DEVICE%, remember, have been assigned already).

```
1070 CTRL% = 0
1080 STAT% = 0
1090 LOVOLT% = 0
1100 HIVOLT% = 4095
1110 LOCHAN% = 0
1120 HICHAN% = 1
```

The first call to AOUS sets the output voltage at D/A channel 0 (LOCHAN%) to −5 Volts (LOVOLT%).

```
1140 CALL AOUS (ADAPT%, DEVICE%, LOCHAN%,
CTRL%, LOVOLT%, STAT%)
```

Note: You must set CTRL% (the expansion device control argument) to 0 when accessing the on-board devices.

Following this call (and all calls to the adapter) functions take the time to execute a test and conditional jump to a brief error-handling routine.

The program sets the execution status variable (STAT%) to 0 before each call.Immediately after each call, it tests STAT%. If STAT% is non-zero, it then sets the variable LNUM% equal to the line number in which the error occurred. The error handler then comes into play, as shown below:

```
1160 IF STAT% <> 0 THEN LNUM% = 1140 : GOTO 6000
```

The error handler looks like this:

```
6000 '****** Error handler begins here *******
6010 '
6020 'on error, print message, number, and exit
6030 PRINT "Execution Error # ";STAT%; "in line
number "; LNUM%
6040 PRINT "Program Terminated'
6050 END
```

If an error occurs, the line and error numbers print, enabling you to determine from the status code what went wrong.

The program then continues, setting D/A channel 1 (HICHAN%) to +5 Volts (HIVOLT%).

```
1180 CALL AOUS (ADAPT%, DEVICE%, HICHAN%, CTRL%,
HIVOLT%, STAT%)
```

After testing STAT% again, the program prints a message on the screen and waits for you to press the Enter key.

```
1230 PRINT "Output voltages set."
1240 PRINT "D/A 0 terminal is −5 Volts"
1250 PRINT "D/A 1 terminal is +5 Volts"
1260 '
1270 '
1280 INPUT "PRESS ENTER TO RUN SECTION 2", C$
1290 CLS
1300 '
```

When you press the Enter key, the screen clears and Section 2 begins.

Section 2: A Simple Analog Input from a Single Channel

Section 2 uses AINS (Analog Input Simple) in a loop to continuously read channel 1 of the on-board analog input device of adapter 0. Channel 1 connects through the potentiometer to +/− 5 volts. By twisting the shaft of the potentiometer, you can modulate the amount of the +5 volt signal that reaches analog input channel 1.

As in the section 1, the first step is to assign values for the arguments of the function.

```
2060 CHANNEL% = 1
2070 CTRL% = 0
2080 STAT% = 0
```

> **Note:** Economical programming standards probably dictate a simple reuse of the variable HICHAN% for the channel number argument. However, the program explicitly re-specifies a CHANNEL% variable here; this emphasizes that all variables used by the CALL statement must be assigned before the program performs the call.

The actual sampling takes place under the control of a WHILE...WEND structure. This takes a sample, displays it as a voltage, and checks for a halt request from the keyboard. As long as no key has been pressed, the adapter samples channel 1 continuously.

```
2100 HALT$ = ""
2110 LOCATE 25,1
2120 PRINT "Press any key to stop sampling."
2130 WHILE HALT$ = ""
2140 'get a sample
2150 CALL AINS (ADAPT%, DEVICE%, CHANNEL%, CTRL%,
V%, STAT%)
2160 'if status non-zero, set line and go
to error handler
2170 IF STAT% <> 0 THEN LNUM% 2150 : GOTO 6000
2180 'convert raw A/D value to volts
2190 VOLTS = V%/409.6-5
2200 'print time of day
2210 LOCATE 1,1,0
2220 PRINT TIME$
2230 'print voltage
2240 LOCATE 4,1,0
2250 PRINT USING "Channel 1 input
voltage is ##.##; VOLTS
2260 LOCATE 4,27,0
2270 'test for halt request
2280 HALT$ = INKEY$
2290 WEND
```

The conversion-to-volts equation in line 2190 converts the 12-bit values (in the range 0 to 4095) returned by the analog input device. It becomes a representation of input voltage in the range +/− 5 Volts. If the analog input system were configured to a different range, you would have to change this equation accordingly.

The general form of this equation is:

```
VOLTAGE = ((CODE * RANGE/2n)- OFFSET)
```

where

- CODE is the raw A/D value.
- RANGE is the A/D.
- Reference voltage span is in volts.
- n is the number of bits of resolution of the A/D converter (exponent of 2).
- OFFSET is the negative offset of the A/D range.A 0 to 10V range, for example, uses the equation VOLTS = V%/409.6 (there is no negative offset). Similarly, if you were using an expansion device of different resolution, you would have to change n to reflect the resolution of that device.

During execution of this program section you turn the shaft of the potentiometer from one stop to the next, varing the fraction of the +/−5V signal supplied to analog input channel 1. The changing voltage is displayed on the screen.

Section 3: Multiple Analog Inputs From a Single Channel

This program uses AINM to input 640 analog values into a one-dimensional array. Unlike the single-sample (AINS) loop used in Section 2, AINM acquires signals at precisely-timed intervals. Since the input signal measured (the voltage modulated by the pot) does not fluctuate very rapidly, you can use a slow sampling rate of 200 samples-per-second. The first lines of this program assign values to the arguments of AINM.Remember, the program has already DIMensioned the RAWDATA% array to include 640 elements.

```
3060 CHANNEL% = 1
3070 CTRL% = 0
3080 MODE% = 0
3090 STOR% = 0
3100 COUNT = 640
3110 RATE = 200
3120 STAT% = 0
```

In this program, COUNT and RATE are real, rather than integer, variables. They are the only non-integer variables for arguments to the functions.

The variable COUNT must not exceed the amount of storage available in the target array. AINM acquires count sample, so a statement of the form:

```
DIM ARRAY% (COUNT - 1)
```

allocates the correct number of array elements.

Often, timed sampling of this sort works in conjunction with a triggering structure. Otherwise, you run a risk that the function will sample while no data is being generated. In this example, Pressing the Enter key triggers the function.

```
3170 INPUT "Press ENTER to begin sampling.", C$
3180 'beep and print "sampling" message, then do AINM
3190 BEEP : CLS
3200 PRINT "Sampling analog input channel 1..."
3210 CALL AINM (ADAPT%, DEVICE%, CHANLO%, CTRL%,
MODE%, STOR%, COUNT, RATE, RAWDATA%(0), STAT%)
```

When the sampling is started the shaft of the potentiometer back and forth to vary the input signal. Once the program collects 640 data points, it jumps to a graphing routine. The routine plots the data, then waits for you to press Enter before moving to the next section.

```
3240 '
3250 'graph points in RAWDATA%
3260 GOSUB 10000
3270 '
```

Section 4: An Analog Input Scan of Multiple Channels

Section 4 uses the function AINSC (Analog Input Scan) to input 320 analog values from each of two channels into a two-dimensional array. This section is in most respects identical to Section 3. It first executes a brief subroutine that outputs a binary low voltage (around 0 volts) to analog input channel 0. This simply provides a second data value for the scan to record. If connected to analog input channel 0, it "floats" to an unpredictable value. By driving it low with a binary output, you guarantee that it returns a constant value near 0 volts.

A simple GOSUB statement in line 4050 jumps to the following subroutine.

```
7040 BDEVICE% = 8
7050 'use bit 8
7060 BIT% = 8
7070 'set bit output low
7080 LO% = 0
7090 STAT% = 0
7100 CALL BITOUS (ADAPT%, BDEVICE%, BIT%, LO%, STAT%)
7110 IF STAT% <> 0 THEN LNUM% = 7037 : GOTO 6000
7120 RETURN
```

This subroutine sets up the required arguments for BITOUS (binary Bit Output Simple); it sets the value of bit 8 to 0. Note that you must use a different variable name, as well as a different number, to specify the device. Simply re-assigning DEVICE%, causes subsequent analog I/O calls to attempt to access device 8. This condition results in an error indication being returned in the status variable.

Next, the program assigns values to the arguments of AINSC. Note that there are two channel arguments: CL%, specifying the low channel (the first channel in the scan) and CH%, specifying the high channel (the last channel in the scan). For each scan specified by the COUNT, AINSC inputs a sample from every channel in the range CL% to CH%, starting with CL% and ending after CH%. In this case, each scan consists of an input from channel 0 followed by an input from channel 1.

```
4080 CL% = 0
4090 CH% = 1
4100 CTRL% = 0
4110 MODE% = 0
4120 STOR% = 0
```

Since each scan can acquire multiple samples, be sure to reserve enough array space for the data by a scanning input function such as AINSC. When DIMensioning arrays filled from scans, you may wish to use a statement of the type:

```
DIM ARRAY% (COUNT*((CH — CL)+1))
```

In the case of this section, you'll find it most convenient to simply reduce the count.

```
4150 COUNT = 320
4160 RATE = 200
4170 STAT% = 0
4180 '
```

This section uses a triggering structure identical to the one used in Section 3. The only difference is in the data acquisition function itself.

```
4210 INPUT "Press ENTER to begin scan.", A$
4220 'beep and print "scanning" message, then
     do AINSC
4230 BEEP: CLS
4240 PRINT "Scanning analog inputs 0 and 1..."
4250 CALL AINSC (ADAPT%, DEVICE%, CL%, CH%, CTRL%,
MODE%, STOR%, COUNT, RATE RAWDATA%(0), STAT%)
```

As in Section 3, press Enter and turn the shaft of the potentiometer. After AINSC has input 320 scans, the program jumps to the plotting subroutine and displays the data.

```
4280 '
4290 'display data
4300 GOSUB 10000
4310 '
```

Section 5: Multiple Analog Outputs From a Single Channel

The final section uses the function AOUM (Analog Output Multiple) to reverse the data acquisition process. This outputs the contents of an array (RAWDATA%), via D/A channel 1, as a varying voltage — an analog signal. When you run this section, the contents of RAWDATA% are output to analog output channel 0. The LED, connected from D/A channel 0 to analog ground (AGND), glows with a varying intensity as the output voltage changes.

The arguments of AOUM first assume these values:

```
5070 CHANNEL% = 1
5080 CTRL% = 0
5090 MODE% = 0
5100 STOR% = 0
5110 COUNT = 640
5120 RATE = 200
5130 STAT% = 0
```

Now print a message and call AOUM.

```
5160 PRINT "Outputting data acquired in Section 4."
5170 '
5180 'and do AOUM
5190 CALL AOUM (ADAPT%, DEVICE%, CHANNEL%, CTRL%,
MODE%, STOR%, COUNT, RATE, RAWDATA%(0), STAT%)
```

BASIC Example Program Listing

```
1 ' PROGRAM: BASIC Example Program for the
2 '          IBM Data Acquisition And Control Adapter
3 '
100 'initialize global variables used in all sections
110 'adapter number 0
120 ADAPT% = 0
140 'use on-board analog I/O device
150 DEVICE% = 9
160 'and DIMension a 640-point array
170 'to be used in sections 3,4, and 5
180 DIM RAWDATA% (639)
190 '
200 ' clear the screen and run the first section
210 CLS
215 KEY OFF
220 INPUT "PRESS ENTER TO RUN SECTION 1", C$
300 '
1000 '***** Section 1 *******
1010 'this routine sets analog output channel #1
1020 'to +5 Volts and analog output channel #2
1030 'to -5 Volts.
1040 '
1050 'assign values to the arguments of AOUS
1060 '(ADAPT% and DEVICE% were initialized above)
1070 CTRL% = 0
1080 STAT% = 0
1090 LOVOLT% = 0
1100 HIVOLT% = 4095
1110 LOCHAN% = 0
1120 HICHAN% = 1
1130 'set voltage to -5 at channel 0
1140 CALL AOUS (ADAPT%, DEVICE%, LOCHAN%, CTRL%,
LOVOLT%, STAT%)
1150 'if status non-zero, set line and go to error handler
1160 IF STAT% <> 0 THEN LNUM% = 1140 : GOTO 6000
1170 'set voltage to +5 at channel 1
1180 CALL AOUS (ADAPT%, DEVICE%, HICHAN%, CTRL%,
HIVOLT%, STAT%)
1190 'if status non-zero, set line and go to error handler
1200 IF STAT% <> 0 THEN LNUM% = 1180 : GOTO 6000
1210 '
1220 'print message
1230 PRINT "Output voltages set."
1240 PRINT "D/A 0 terminal is -5 Volts"
1250 PRINT "D/A 1 terminal is +5 Volts"
1260 '
1270 '
1280 INPUT "PRESS ENTER TO RUN SECTION 2", C$
1290 CLS
1300 '
2000 '******* Section 2 *************
2010 'this routine continuously inputs and
2020 'displays as a voltage the analog value
2030 'from channel 1 of the on-board device
2040 '
2050 'assign values for the arguments of AINS
2060 CHANNEL% = 1
2070 CTRL% = 0
2080 STAT% = 0
2090 ' then set up for the loop
2100 HALT$ = ""
2110 LOCATE 25,1
2120 PRINT "Press any key to stop sampling."
2130 WHILE HALT$ = ""
2140 'get a sample
2150 CALL AINS (ADAPT%, DEVICE%, CHANNEL%, CTRL%,
V%, STAT%)
2160 'if status non-zero, set line and go to error handler
2170 IF STAT% <> 0 THEN LNUM% = 2150 : GOTO 6000
```

```
2180 'convert raw A/D value to volts
2190 VOLTS V%/409.6-5
2200 'print time of day
2210 LOCATE 1,1,0
2220 PRINT TIME$
2230 'print voltage
2240 LOCATE 4,1,0
2250 PRINT USING "Channel 1 input
voltage is ##.##"; VOLTS
2260 LOCATE 4,27,0
2270 'test for halt request
2280 HALT$ =NKEY$
2290 WEND
2300 CLS
2310 '
2320 INPUT "PRESS ENTER TO RUN SECTION 3", C$
2330 CLS
2340 '
3000 '********** Section 3 **************
3010 'this program takes 640 samples from
3020 'channel 1 of the on-board analog input device
3030 'and stores them in a one-dimensional array
3040 '
3050 'assign values to the arguments of AINM
3060 CHANNEL% = 1
3070 CTRL% = 0
3080 MODE% = 0
3090 STOR% = 0
3100 COUNT = 640
3110 RATE = 200
3120 STAT% = 0
3130 '
3140 '
3150 'set up a structure that allows the user to
3160 'start sampling by pressing the ENTER key
3170 INPUT "Press ENTER to begin sampling.", C$
3180 'beep and print "sampling" message, then do AINM
3190 BEEP : CLS
3200 PRINT "Sampling analog input channel 1..."
3210 CALL AINM (ADAPT%, DEVICE%, CHANNEL%, CTRL%,
MODE%, STOR%, COUNT, RATE, RAWDATA%(0), STAT%)
3220 'if status non-zero, set line and go to error handler
3230 IF STAT% <> 0 THEN LNUM% = 3210 : GOTO 6000
3240 '
3250 'graph points in RAWDATA%
3260 GOSUB 10000
3270 '
3280 '
3290 '
3300 '
3310 INPUT "PRESS ENTER TO RUN SECTION 4", C$
3320 SCREEN 0, 0, 0
3330 WIDTH 80
3340 CLS
3400 '
4000 '***** Section 4 *****
4010 'this routine takes 50 samples each from
4020 'channels 0 and 1 of the analog input device
4030 'and stores them in a two-dimensional array
4040 'Start by driving analog input 0 low
4050 GOSUB 7000
4060 '
4070 'assign values to the arguments of AINSC
4080 CL% = 0
4090 CH% = 1
4100 CTRL% = 0
4110 MODE% = 0
4120 STOR% = 0
4130 'note that the count must be 320, rather than 640,
4140 'since each scan acquires two samples
4150 COUNT = 320
4160 RATE = 200
4170 STAT% = 0
4180 '
4190 'set up a structure that allows the user to
4200 'start sampling by pressing the ENTER key
4210 INPUT "Press ENTER to begin scan.", A$
4220 'beep and print "scanning" message, then do AINSC
4230 BEEP: CLS
4240 PRINT "Scanning analog inputs 0 and 1..."
4250 CALL AINSC (ADAPT%, DEVICE%, CL%, CH%, CTRL%,
MODE%, STOR%, COUNT, RATE, RAWDATA%(0), STAT%)
4260 'if status non-zero, set line and go to error handler
4270 IF STAT% <> 0 THEN LNUM% = 4250 : GOTO 6000
4280 '
4290 'display data
4300 GOSUB 10000
4310 '
4320 INPUT "PRESS ENTER KEY TO RUN SECTION 5", C$
4330 SCREEN 0, 0, 0
4340 WIDTH 80
4350 CLS
4360 '
5000 '********Section 5************
5010 ' This example outputs the contents of
5020 ' RAWDATA% to the analog output channel #1
5030 ' An LED connected from channel 1 to channel 0
5040 ' changes intensity as the output voltage changes
5050 '
5060 'assign values to the arguments of AOUM
5070 CHANNEL% = 0
5080 CTRL% = 0
5090 MODE% =
5100 STOR% = 0
5110 COUNT = 640
5120 RATE = 200
5130 STAT% = 0
5140 '
5150 'print message
5160 PRINT "Outputting data acquired in Section 4."
5170 '
5180 'and do AOUM
5190 CALL AOUM (ADAPT%, DEVICE%, CHANNEL%, CTRL%,
MODE%, STOR%, COUNT, RATE,.RAWDATA%(0), STAT%)
5200 'if status non-zero, set line and go to error handler
5210 IF STAT% <> 0 THEN LNUM% = 5190 : GOTO 6000
5220 LOCATE 10,1,0
5230 PRINT "End of the Sample Program."
5240 END
5250 '
5260 '
5500 '
```

```
6000 '****** Error handler begins here *******
6010 '
6020 'on status error, print message, error number, and exit
6030 PRINT "Execution Error # ";STAT%; "in line
number "; LNUM%
6040 PRINT "Program Terminated"
6050 END
6100 '
7000 '******* Binary output subroutine begins here ********
7010 'Set up to drive analog input 0 low using BITOUS
7020 'initialize variables
7030 'use binary device 8
7040 BDEVICE% = 8
7050 'use bit 8
7060 BIT% = 8
7070 'set bit output low
7080 LO = 0
7090 STAT% = 0
7100 CALL BITOUS (ADAPT%, BDEVICE%, BIT%, LO%, STAT%)
7105 'if status non-zero, set line and go to error handler
7110 IF STAT% <> 0 THEN LNUM% = 7037 : GOTO 6000
7120 RETURN
8000 '
9000 ' ********* Plotting Routine begins here ************
9999 '
10000 ' Name: SIMPLOT
10010 '
10020 ' Arguments: LENGTH% = number of points to plot
10030 '             RAWDATA%(.) = array to be plotted
10040 ' Affects: The screen is left in mode 2. All
10050 '           variables ending in .GRF are reserved.
10060 '
10070 ' This subroutine plots the elements of the array
10080 ' RAWDATA%(.) on a color display. The elements are
10090 ' windowed to the range specified by MAXVAL.GRF and
10100 ' MINVAL.GRF. THe number of points plotted must be
10110 ' passed as an argument in the variable LENGTH%.
10120 ' The screen clears upon each execution of SIMPLOT.
10130 '
20000 MAXVAL.GRF = 4095    ' maximum value to be plotted
20005 LENGTH% = 640
20010 MINVAL.GRF = 0       'minimum value to be plotted
20020 '
20030 SCREEN 2
20040 CLS
20050 '
20060 VIEW (100,50) - (600,150),,1
20070 WINDOW (0, MINVAL.GRF) - (LENGTH% - 1, MAXVAL.GRF)
20080 '
20090 FOR I.GRF% = 0 TO LENGTH% - 1
20100 PSET (I.GRF%, RAWDATA%(I.GRF%))
20110 NEXT I.GRF%
20120 '
20130 RETURN
```

IBM General Purpose Interface Bus Adapter and Software

IBM GPIB Adapter and Programming Support

Introduction

The IBM Personal Computer General Purpose Interface Bus Adapter and the Programming Support products enable engineering and science professionals to use IBM Personal Computers IBM to access and control over 2,000 different instruments or devices designed to the IEEE-488 standard. Each personal computer can accommodate up to four GPIB Adapters and provide software support for up to 48 devices.

The IBM PC, equipped with the GPIB Adapter, cable, and programming support software can be a talker, listener, or controller. A talker is able to send data, a listener is able to receive data, and a controller can address and trigger specific instruments as well as perform bus management functions. The GPIB Adapter also provides capabilities for cost-effective data transfer between workstations, and the connection of several computers for sharing instruments or peripheral I/O devices.

GPIB Adapter Features

IBM PC GPIB Adapter features include:

- Half-size card that can be conveniently installed: even in an IBM Portable Personal Computer.
- FCC Class B certification.
- Up to 20K bytes/second programmed I/O data rate.
- Up to 300K bytes/second programmed DMA data rate.
- Complete IEEE-488 controller, talker and listener capability.
- Adapter designed to the IEEE-488 Standard, 1975/78, including the 488A-1980 supplement.
- Up to 14 devices or instruments on the GPIB.
- User-selectable Direct Memory Access channel for enhanced performance.
- User-selectable interrupt level.
- Extensive diagnostics provided to aid in hardware configuration and problem determination.

Adapter Functional Description

The IBM GPIB Adapter can be thought of as a bus translator, converting messages and signals present on the IBM PC I/O Channel into appropriate GPIB messages and signals. Expressed in IEEE-488 terminology, the adapter implements GPIB interface functions for communicating with the IBM PC central processor and memory. From the point of view of the IBM PC, the GPIB Adapter is an interface to GPIB devices.

The GPIB Adapter bus interface consists of the following major functions:

- Address Decoding
- Buffering and Data Routing
- Interrupt Arbitration
- DMA Arbitration
- Configuration Switches/Jumpers
- GPIB Adapter logic — Talker/Listener/Controller

ADDRESS DECODING monitors the address lines to recognize when the GPIB I/O address is present on the IBM PC I/O channel and enables read and write access to the GPIB Adapter.

BUFFERING AND DATA ROUTING handles data transfer between the IBM PC I/O Channel and the GPIB Adapter through a bidirectional internal data bus.

INTERRUPT ARBITRATION recognizes when interrupts have been enabled or disabled and passes or inhibits them accordingly.

DMA ARBITRATION recognizes when DMA operations are enabled or disabled, and when the last transfer has taken place, it also routes the DMA request and acknowledge signals to the selected DMA channels.

CONFIGURATION SWITCHES/JUMPERS determine the I/O port address, the DMA channel pair used, and the interrupt request line used.

The UPD7210 GPIB Adapter logic implements virtually all of the IEEE-488 interface functions to interact with other devices on the GPIB. Within the GPIB Adapter are 21 program registers which are used to configure, control, and monitor the interface functions and to pass commands and data to and from the GPIB. Special-purpose, multi-function transceivers interface the GPIB Adapter to the GPIB.

The capabilities of the GPIB Adapter below are in terms of the codes in Appendix C of the IEEE-488 Standard.

GPIB Interface Capabilities

IEEE Code	Description
• SH1	Complete source handshake capability.
• AH1	Complete acceptor handshake capability DAC and RFD holdoff on certain events.
• T5	Complete talker capability: Basic talker Serial poll Talk only mode Unaddressed on MLA Send END or EOS Dual primary addressing
• TE5	Complete extended talker capability: Basic extended talker Serial poll Talk only mode Unaddressed on MSA and TPAS Send END or EOS
• L3	Complete listener capability: Basic listener Listen only mode Unaddressed on MTA Detect END or EOS Dual primary addressing
• LE3	Complete extended listener capability: Basic listener Listen only mode Unaddressed on MSA and LPAS Detect END or EOS
• SR1	Complete service request capability.
• PP1	Remote parellel poll configuration.
• C1-5	Complete controller capability: System controller Send IFC and take control Send REN Respond to service request Send interface messages Receive control Pass control Parallel poll Take control synchronously or asynchronously.
• E1/2	Three-state bus driver with automatic switch to open collector with parallel poll.

The GPIB Adapter has complete source and acceptor handshake capability. The Adapter can operate as a basic talker or extended talker and can respond to a serial poll. It becomes unaddressed to talk when it receives its listen address. The interface can operate as a basic listener or extended listener. It becomes unaddressed to listen when it receives its talk address. The GPIB Adapter has full capabilities for requesting service from another controller. The ability to place the GPIB Adapter in local mode is included but the interpretation of remote versus local mode is software dependent. Full parallel poll capability is included into the interface. Device clear and trigger capability is included in the interface but the interpretaion is software dependent. All controller functions as specified by the IEEE-488 standard are included in the adapter. These include the capability to:

- Be programmed to be the system controller for the purposes of initializing the GPIB interfaces of other devices and placing those devices in remote or local program mode. (It may also be programmed not to be the system controller so that the IBM PC can be used on a GPIB with another controller which is the system controller.)
- Send multiline interface messages or commands to other devices.

- Detect when other devices are requesting service and conduct serial or parallel polls.
- Pass control to other controllers and receive control from them.
- Go to standby to allow the addressed talker to send data to addressed listeners, and to take control again synchronously or asynchronously.

Dimensions

Length: 114.30mm (4.500in. ± .002)
Height: 106.70mm (4.200in. ± .002)

Bus connection

An IEEE-488 compatible cable is required and is not a part of the IBM Personal Computer GPIB Adapter offering.

Operating environment

Operating temperatures: +15.6 to +32.2 C (+60 to +90 F) Relative humidity: 8 to 80% (non-condensing) Altitude: 2187m (7000 ft.) maximum.

Requirements

The IBM Personal Computer GPIB Adapter may be used with and IBM Personal Computer AT, IBM PC XT, IBM PC, or IBM Portable PC. The IBM PC GPIB Adapter Programming Support requires the IBM Disk Operation System (DOS) 2.00 or higher, and the IBM PC GPIB Adapter. The IBM Personal Computer AT requires DOS 3.00.

PC interface

The adapter connects to the 62-pin IBM PC bus and is I/O mapped. It supports both DMA and Interrupts.

Performance

Programmed I/O data rate up to: 20K bytes/second DMA data rate up to: 300K bytes/second Number of device loads: 14 (maximum)

Power

Power requirements: 5 volts DC ± 5%
Current: 450 milliamps (typical)
Power: 2.25 watts

General Purpose Interface Bus Programming Support

Programming support features

The IBM PC GPIB Adapter Programming Support offers you a flexible way to control large instrument and measurement systems accurately and productively. The software includes a loadable device driver, interactive configuration and control programs and a high-level language subroutine library to support the adapter. IBM PC GPIB Adapter software features include:

- Menu-driven interactive configuration program for easy software/hardware configuration.
- Interactive control program that permits user control and monitoring of devices for immediate verification of a selected configuration.
- High-level functions for direct device control that make GPIB management complexities transparent to the user.
- Low-level functions for direct application software control of all adapter interface commands.
- Support for application development in Interpretive BASICA, Compiled BASIC, FORTRAN 2.00, IBM PC Professional FORTRAN, and Lattice C languages.
- Use of symbolic device names.
- Two device drivers: one supports two adapters and a total of 16 devices; the other supports four adapters and a total of 48 devices.
- Examples of programming techniques for GPIB functions, and sample programs for each supported language.

High and Low Level Functions

The Programming Support software provides a comprehensive set of high level and low level functions for communicating with and controlling devices on the GPIB. The high level device related functions automatically execute all of the several operations needed to perform certain activities on the GPIB, like reading from and writing to devices and polling them for status. These functions are designed to free the user from having to know the specific GPIB protocol or bus management details involved in specific operations. They are called high level because they are easy to learn and to use. Low level adapter related functions, by contrast, perform rudimentary or primitive operations that generally require more extensive knowledge of GPIB protocol in order to be used properly. They are needed, however, because the high level functions may not be sufficient in all applications. In these cases, the low level functions provide flexibility and versatility in controlling the GPIB so that any application problem can be solved.

GPIB Software Functions

FUNCTION	DESCRIPTION
IBBNA	Specify device access adapter
IBCAC	Reactivate controller
IBCLR	Clear device
IBCMD	Send commands
IBCMDA	Send commands asynchronously
IBDMA	Enable or disable DMA
IBEOS	Change or disable EOS termination method
IBEOT	Enable or disable end termination message
IBFIND	Return adapter or device unit descriptor
IBGTS	Active controller to standby
IBIST	Sets or clears individual status bit
IBLOC	Local mode
IBONL	Place device on or offline
IBPAD	Change primary address
IBPCT	Pass GPIB controller-in-charge to a device
IBPPC	Parallel poll configure
IBRD	Read data
IBRDA	Read data asynchronously
IBRDF	Read data from GPIB into file
IBRPP	Conduct a parallel poll
IBRSC	Request or release system control
IBRSP	Conduct serial poll
IBRSV	Request service from the GPIB controller-in-charge
IBSAD	Change or disable secondary address
IBSIC	GPIB interface clear
IBSRE	Set or clear remote enable line
IBSTOP	Stop asynchronous operation
IBTMO	Change or disable a device timeout limit
IBTRG	Trigger device
IBWAIT	Wait for selected event
IBWRT	Write data
IBWRTA	Write data asynchronously
IBWRTF	Write data from file

GPIB Programming Support Utility Programs

Two utility programs are provided to aid the user in bringing up a system. These programs are:

- The GPIB Configuration Program
- The Interface Bus Interactive Control Program

The GPIB Configuration Program

The purpose of the configuration program is to describe all of the devices and adapters in the system so that the user does not have to remember their key features or characteristics when writing an application program. The configuration program is used to pass the key features of the GPIB devices to the device support software. These features include each device's GPIB address, read and write termination mode, I/O timeout limit, and controlling GPIB Adapter. In the application program, it is only necessary to refer to the device by a symbolic name, and the device driver takes care of the detail. The following list of characteristics are passed to the device driver via the configuration program. The first group are characteristics of devices. The second group are adapter related.

Device Characteristics

- The symbolic name of each device on the GPIB (such as "VMETER").
- The access adapter for each device.
- The primary and, if used, the secondary address for each device.
- The time limit that is to be imposed when executing device I/O functions.
- How I/O transmissions to and from the device terminate.

Adapter Characteristics

- The symbolic name of each adapter in the system.
- The adapter's computer I/O or port address.
- If the adapter is the system controller of the devices on its bus.
- The time limit during execution of adapter I/O functions.

- How I/O transmissions to and from the adapter terminate.
- What DMA channel, if any, the adapter uses.
- Whether it uses high speed or normal timing when transmitting data to a device.

Once this information has been identified and passed to the device driver specified using the GPIB Configuration Program, the device driver may be installed in the operating system.

The Interface Bus Interactive Control Program

The interactive control program allows the user to perform any function (reading, writing, polling, etc.) from the terminal. The program can control the GPIB devices independent of an application program, providing the user with a powerful software development and troubleshooting tool. It can be used to learn how to program GPIB devices, to measure how system performance varies with different program sequences, to debug an application program one step at a time, or to locate a malfunctioning device on the GPIB.

The Interface Bus Interactive Control Program allows the user to learn about the GPIB software, his instrument, about his system, and trouble shoot problems on the bus. The software is designed with built in error checking and where appropriate checks on the validity of arguments that the user passes. The user may make run-time function calls to change the configuration values such as a device's primary address. Errors are detected quickly and are reported as soon as they occur. This means that the user does not propagate the error until some situation is reached where he cannot decipher what is going on. Built into the software is time-out support. Unless specifically requested by the user not to, the software will time-out certain operations on the bus so that there is no case where the processor or the bus will hang-up indefinitely.

Example Programs

Two GPIB example programs written in Interpretive BASICA are listed below. The first uses low level adapter function calls and the second uses high level device function calls. The primary objective of each example is to acquire a voltage reading from the digital volt meter DVM. These examples demonstrate GPIB programming techniques using selected functions supported by the IBM GPIB Adapter Programming Support software.

BASICA Example Program (Adapter)

This sample program illustrates adapter level programming techniques using the BASICA language interface.

```
100 REM IBM BASIC - Using adapter function calls
101 REM Merge this code with the BASICA header
102 REM
110     ADNAME$ "GPIB0"
115 REM
120 REM Assign a unique identifier to adapter 0
130 REM and store in variable GPIB0%
135 REM
140     CALL IBFIND (ADNAME$,GPIB0%)
145 REM
150 REM Check for error on IBFIND call
152 REM
155     IF GPIB0% < 0 GOTO 2000
160 REM
165 REM Send the IFC (Interface Clear) message
170 REM to all devices.
175 REM
180     CALL IBSIC (GPIB0%)
190 REM
200 REM Check for an error on each GPIB call
210 REM to be safe.
215 REM
220     IF IBSTA% < 0 THEN GOTO 3000
230 REM
240 REM Turn on the REN (Remote Enable) signal.
245 REM
250     V% = 1 : CALL IBSRE (GPIB0%,V%)
260     IF IBSTA% < 0 THEN GOTO 3000
270 REM
280 REM Inhibit from panel control with the LLO
290 REM (local lockout) command, place the DVM
300 REM in remote mode by addressing it to
310 REM listen, send the DCL (device clear)
320 REM message to clear internal device
330 REM functions, and address the GPIB Adapter to
340 REM talk.
345 REM
350     CMD$ = CHR$(&H11)+ "#" +CHR$ (&H14)+ "@"
360     CALL IBCMD (GPIB0%,CMD$)
370     IF IBSTA% < 0 THEN GOTO 3000
380 REM
390 REM Write the function, range, and trigger
400 REM source instructions to the DVM.
405 REM
410     WRT$ = "F3R7T3" : CALL IBWRT (GPIB0%,WRT$)
420     IF IBSTA% < 0 THEN GOTO 3000
430 REM
440 REM Send the GET message to trigger a
450 REM measurement reading.
455 REM
460     CMD$ = CHR$(&H8) : CALL IBCMD (GPIB0%,CMD$)
470     IF IBSTA% < 0 THEN GOTO 3000
480 REM
490 REM Wait for the DVM to set SRQ or for a
500 REM timeout; if the current timeout limit
```

```
510 REM is too short, use IBTMO to change it.
515 REM
520     MASK% = &H5000
525     CALL IBWAIT (GPIB0%, MASK%)
530     IF (IBSTA% AND &HC000) <> 0 GOTO 3000
540 REM
550 REM Since neither a timeout nor an error
560 REM occurred, IBWAIT must have returned
570 REM on SRQ. Next do a serial poll.
580 REM First unaddress bus devices and send
590 REM the SPE (Serial Poll Enable) command,
600 REM then send the DVM's talk address and
610 REM the GPIB Adapter listen address (ASCII
620 REM space).
625 REM
630     CMD$ = "?_" + CHR$(&H18) + "C "
640     CALL IBCMD (GPIB0%,CMD$)
650     IF IBSTA% < 0 THEN GOTO 3000
660 REM
670 REM Now read the status byte. If it is
680 REM &HC0, the DVM has valid data to send;
690 REM otherwise, it has a fault condition
700 REM to report.
705 REM
710     RD$ = SPACE$(1) : CALL IBRD (GPIB0%,RD$)
720     IF IBSTA% < 0 THEN GOTO 3000
730     IF ASC(RD$) <> &HC0 THEN GOTO 4000
740 REM
750 REM Complete the serial poll by sending
760 REM the SPD (Serial Poll Disable) message.
765 REM
770     CMD$ = CHR$(&H19) : CALL IBCMD (GPIB0%,CMD$)
780     IF IBSTA% < 0 THEN GOTO 3000
790 REM
800 REM Since the DVM and GPIB Adapter are
810 REM still addressed to talk and listen,
820 REM the measurement can be read as follows.
825 REM
830     RD$ = SPACE$(16) : CALL IBRD (GPIB0%,RD$)
840     IF IBSTA% < 0 THEN GOTO 3000
850 REM
860 REM To close out a programming sequence,
870 REM send IFC to initialize the bus and
880 REM call the IBONL function to place the
890 REM GPIB Adapter online.
895 REM
900     CALL IBSIC (GPIB0%)
910     V% = 0 : CALL IBONL (GPIB0%, V%)

2000     REM A routine at this location would
2010     REM notify the user that the IBFIND
2020     REM call failed, and refer him to
2030     REM the driver software configuration
2040     REM procedures.

3000     REM An error checking routine at this
3010     REM location would, among other things,
3020     REM check IBERR to determine the exact
3030     REM cause of the error condition and
3040     REM then take action appropriate to
3050     REM the application. For errors during
3060     REM data transfers, IBCNT may be
3070     REM examined to determine the actual
3080     REM number of bytes transferred.
4000     REM A routine at this location would analyze
4010     REM the fault code returned in the DVM's
4020     REM status byte and take appropriate
4030     REM action.
4040     STOP
5000     END
```

BASICA Example Program (Device)

This sample program illustrates device level programming techniques using the BASICA language interface.

```
100 REM IBM BASIC - Using device function calls
101 REM Merge this code with the BASICA header
110 REM Assign a unique identifier to device
120 REM and store in variable DVM%.
125 REM
130     DEV$ = "DVM"
140     CALL IBFIND (DEV$,DVM%)
145 REM
150 REM Check for error on IBFIND call
155 REM
160     IF DVM% < 0 THEN GOTO 2000
170 REM
180 REM Clear the device
185 REM
190     CALL IBCLR (DVM%)
195 REM
200 REM Check for an error on each GPIB call
210 REM to be safe.
215 REM
220     IF IBSTA% < 0 THEN GOTO 3000
230 REM
240 REM Write the function, range, and trigger
250 REM source instructions to the DVM.
255 REM
260     WRT$ = "F3R7T3" : CALL IBWRT (DVM%,WRT$)
270     IF IBSTA% < 0 THEN GOTO 3000
280 REM
290 REM Trigger the device.
295 REM
300     CALL IBTRG (DVM%)
310     IF IBSTA% < 0 THEN GOTO 3000
320 REM
330 REM Wait for the DVM to set RQS or for a
340 REM timeout; if the current timeout limit
350 REM is too short, use IBTMO to change it.
355 REM
360     MASK% = &H4800 : CALL IBWAIT (DVM%, MASK%)
370     IF (IBSTA% AND &HC000) <> 0 THEN GOTO 3000
380 REM
390 REM Since neither a timeout nor an error
```

```
400 REM occurred, IBWAIT must have returned
410 REM on RQS. Next, serial poll the device.
415 REM
420     CALL IBRSP (DVM%,SPR%)
430     IF IBSTA% < 0 THEN GOTO 3000
440 REM
450 REM Now test the status byte (SPR%).
460 REM If SPR% is &HC0, the DVM has valid
470 REM data to send; otherwise, it has a
480 REM fault condition to report.
485 REM
490     IF SPR% <> &HC0 THEN GOTO 4000
500 REM
510 REM If the data is valid, read the
520 REM measurement.
525 REM
530     RD$ = SPACE$(16) : CALL IBRD (DVM%,RD$)
540     IF IBSTA% < 0 THEN GOTO 3000
550 REM
560 REM To close out a programming sequence,
570 REM reset the device.
575 REM
580     CALL IBCLR (DVM%)
2000    REM A routine at this location would
2010    REM notify the user that the IBFIND
2020    REM call failed, and refer him to
2030    REM the driver software configuration
2040    REM procedures.
3000    REM An error checking routine at this
3010    REM location would, among other things,
3020    REM check IBERR to determine the exact
3030    REM cause of the error condition and
3040    REM then take action appropriate to
3050    REM the application. For errors during
3060    REM data transfers, IBCNT may be
3070    REM examined to determine the actual
3080    REM number of bytes transferred.
4000    REM A routine at this location would analyze
4010    REM the fault code returned in the DVM's
4020    REM status byte and take appropriate
4030    REM action.
4040    STOP
5000    END
```

IBM Corporation
Editor, IBM Personal Computer Seminar Proceedings
4629
Post Office Box 1328
Boca Raton FL 33432