A COMMUNICATION PLATFORM
FOR DISTRIBUTED PC/MAINFRAME APPLICATIONS
WITHIN A 3270 ENVIRONMENT

by

Stephen Gibbs Bears

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of
Master of Science
in
Electrical Engineering

APPROVED:

[Signatures]

Dr. Charles E. Nunnally, Chairman

[Signatures]

Dr. Joseph G. Tront

Dr. Scott F. Midkiff

February 1992

Blacksburg, Virginia
A COMMUNICATION PLATFORM
FOR DISTRIBUTED PC/MAINFRAME APPLICATIONS
WITHIN A 3270 ENVIRONMENT

by

Stephen Gibbs Bears
Dr. Charles E. Nunnally, Chairman

(ABSTRACT)

Remote personal computer communication with IBM mainframes is often confined to low throughput (less than 19,200 baud), asynchronous serial lines managed by the mainframe through 3270 protocol converters. The capabilities of the personal computer are under utilized and limited to terminal emulation and file transfer. For such an environment, a software solution is presented to improve the computing platform between IBM mainframes and personal computers without modifying any existing, intervening communication equipment. Transparent communication support for distributed, interactive applications is provided through the operation of a data link control protocol. The communication services are applied to the development of a distributed WYSIWYG page previewer for SCRIPT/VS.
Acknowledgements

I declare that the Lord Jesus Christ is righteous and faithful (Psalm 121)! I acknowledge that in all ways God has been faithful to me (Deut 2:7). All blessing and honor be to God the Father, God the Son, and God the Holy Spirit forever and ever. Amen.

I especially thank my parents whose prayers, love, and encouragement have always been present. I thank the Lord for blessing me with such wonderful parents. May you continue to grow in the wisdom and love of the Lord.

I thank Ray Stell for being a friendly source of information which made the development of this project easier. Much thanks to all my extended family in the Johnson house group for their prayer support. Thanks should also be extended to Dr. Nunnally for serving as my advisor and giving strong direction to this thesis. May the Lord bless you with salvation along with Dr. Midkiff and Dr. Tront.

And last, but certainly in no sense the least, I thank my friends Wayne and Ron for their assistance and support. May the Lord continue to lead and bless you in all your endeavors.

Those who trust in the Lord shall never be disappointed.
# Table of Contents

1.0 Introduction .......................................................... 1
   1.1 PC/Mainframe Characteristics .................................. 3
   1.2 Existing 3270 Support .......................................... 5
   1.3 Extending 3270 Support ......................................... 7
   1.4 Distributed Page Previewer ..................................... 9

2.0 Communication Platform ........................................... 11
   2.1 Communication Module ......................................... 14
      2.1.1 DLC Protocol Description .............................. 17
      2.1.2 DLC Protocol Implementation .......................... 26
         Local System .............................................. 28
         Remote System ........................................... 31
   2.2 Terminal Emulator/Command Module .......................... 33
   2.3 Database Module ................................................ 35

3.0 Page Previewer Application ..................................... 36
   3.1 Page Previewer Description ................................... 37
   3.2 Application Division ........................................... 41
   3.3 Communication Strategy ....................................... 44
      3.3.1 Development Guidelines ............................... 45
      3.3.2 Page Previewer Organization .......................... 48
   3.4 Physical Compression ........................................... 52
      3.4.1 Run Length Encoding ................................... 56
      3.4.2 Huffman Compression ................................... 58
      3.4.3 Lempel-Ziv-Welch Compression ........................ 60
4.0 Further Development ........................................ 62
  4.1 Communication Platform Extensions .................. 63
  4.2 Page Previewer Extensions .............................. 65
  4.3 Other Applications ...................................... 68

5.0 Conclusions and Recommendations .................... 70

Bibliography .................................................. 73

Appendices .................................................... 76
  A: 7171 Protocol Converter ................................ 76
  B: Page Previewer PMP Commands .......................... 81
  C: Pattern Display Algorithm .............................. 85
  D: Page Previewer Operation ................................ 110

Vita .......................................................... 135
Illustrations

**Figure**

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>Communication Platform Software</td>
<td>12</td>
</tr>
<tr>
<td>2-2</td>
<td>Communication Platform Evolution</td>
<td>13</td>
</tr>
<tr>
<td>2-3</td>
<td>Communication Platform Implementation</td>
<td>13</td>
</tr>
<tr>
<td>2-4</td>
<td>Communication Environment</td>
<td>15</td>
</tr>
<tr>
<td>2-5</td>
<td>Packet Format</td>
<td>20</td>
</tr>
<tr>
<td>2-6</td>
<td>Packet Format Byte</td>
<td>22</td>
</tr>
<tr>
<td>2-7</td>
<td>Communication Module</td>
<td>27</td>
</tr>
<tr>
<td>3-1</td>
<td>Document Printing Process</td>
<td>38</td>
</tr>
<tr>
<td>3-2</td>
<td>Document Previewing Process</td>
<td>40</td>
</tr>
<tr>
<td>3-3</td>
<td>Page Previewer Organization</td>
<td>50</td>
</tr>
<tr>
<td>C-1</td>
<td>Page Orientation Coordinate Systems</td>
<td>86</td>
</tr>
<tr>
<td>C-2</td>
<td>Font Pattern Position Parameters</td>
<td>90</td>
</tr>
<tr>
<td>C-3</td>
<td>Pattern Load Orientations</td>
<td>93</td>
</tr>
<tr>
<td>C-4</td>
<td>Print-point Position Relative to Load Orientation</td>
<td>94</td>
</tr>
<tr>
<td>C-5</td>
<td>Display Window Reference Diagram</td>
<td>97</td>
</tr>
<tr>
<td>C-6</td>
<td>Pattern Positions Within the Display Window</td>
<td>99</td>
</tr>
<tr>
<td>C-7</td>
<td>Row Trimming Algorithm</td>
<td>101</td>
</tr>
<tr>
<td>C-8</td>
<td>Column Trimming Algorithm</td>
<td>102</td>
</tr>
<tr>
<td>C-9</td>
<td>Pel to Pixel Mapping</td>
<td>104</td>
</tr>
<tr>
<td>D-1</td>
<td>Example SCRIPT Document: &quot;Slide&quot;</td>
<td>114</td>
</tr>
<tr>
<td>D-2</td>
<td>Example SCRIPT Document: &quot;Schedule&quot;</td>
<td>122</td>
</tr>
<tr>
<td>D-3</td>
<td>Initial Display Mode</td>
<td>124</td>
</tr>
<tr>
<td>D-4</td>
<td>Scan Page Window of Figure D-3</td>
<td>125</td>
</tr>
<tr>
<td>D-5</td>
<td>Full Page Landscape View</td>
<td>128</td>
</tr>
<tr>
<td>D-6</td>
<td>Scan Page Window of Figure D-5</td>
<td>129</td>
</tr>
<tr>
<td>D-7</td>
<td>Split Screen Mode</td>
<td>131</td>
</tr>
</tbody>
</table>
Table

2-1 Local System Packet Reception 19
2-2 Remote System Packet Reception 19
3-1 Compression Results 54
3-2 PMP File Run Length Frequencies 57
5-1 PMP File Transfer Time at 2400 Baud 71
8-1 PMP Command Support 82
C-1 Orientation Abbreviations 85
C-2 Coordinate System Transformation Equations 88
C-3 Reduced Coordinate Transformation Equations 88
C-4 Print-point Location as a Function of Print Direction 92
C-5 Print-point Location as a Function of Load Orientation 92
C-6 Print-point to Base Coordinate Translation 95
C-7 Boundary Check Inequalities for the Display Window 98
C-8 Compressed Boundary Check Inequalities 98
C-9 Boundary Check Constants 99
C-10 Row Trimming Algorithm Variable Initialization 101
C-11 Column Trimming Algorithm Variable Initialization 102
C-12 Starting Row and Column Coordinates 105
C-13 Row Adjustment Algorithms 105
C-14 Row and Column Redefinition 107
C-15 Plot Routine Pixel Location 107
C-16 End Point Location as a Function of Print Direction 109
C-17 End Point Location as a Function of Load Orientation 109
CHAPTER 1
INTRODUCTION

In an IBM mainframe environment, the processing and graphics capabilities of remotely located personal computers (PCs) are often under-utilized. Although the PC's processing power does not match that of the mainframe, it still possesses some complementary features which could enhance mainframe access if placed within a distributed computing environment. But initializing a distributed environment is often hampered by the existing communication link, which is often a low throughput (less than 19,200 baud) asynchronous serial line managed by the mainframe through a protocol converter. For such an environment, a software solution is presented to improve the computing platform between IBM mainframes and PCs without modifying any existing, intervening communication equipment. The main thrust of this thesis is to develop a communication platform that may be used to launch distributed, interactive mainframe applications which utilize the PC's graphics and input devices in a more integrated manner. Specifically, a rudimentary data link control (DLC) protocol layer has been implemented over the existing LU 2 communication interface. The DLC provides distributed applications with transparent communication services which are designed to reduce the apparent communication requirements. The idea is to establish support for distributed applications running from remotely located PCs which do not yet have access to the mainframe through an LU 6.2 protocol interface. The communication services which have been developed are grouped together into a communication platform and are presented through the continued development of a SCRIPT "what you see is what you get" (WYSIWYG) page previewer. Additionally, such a platform may be extended to provide virtual services, while still supporting full screen main-

1.0 Introduction
frame applications that require terminal emulation, with less code overhead than a full PC implementation of the LU 6.2 protocol [12].
1.1 PC/Mainframe Characteristics

The mainframe and PC's weaknesses and strengths tend to complement one another. The mainframe represents a system which has tremendous processing power in terms of MIPS and accuracy, is able to do concurrent processing, has vast amounts of storage, and the installed and available software base can be quite powerful. In addition, the mainframe’s place in the corporate setting remains because of the need for businesses to maintain large, secure, and redundant databases which are supported centrally for data consistency. But these capabilities and resources may not be widely used depending upon the access connection and how the mainframe is managed. Users may avoid the mainframe because of the user interface, the access costs (equipment and operating), the lack of available specialized software and tools, or because management decisions cannot be controlled or tailored to suit particular or specialized needs.

PC software and hardware may be considerably less powerful than the mainframe but is often obtainable at a lower price to performance ratio and is a larger selection of vendors and sources. As a single user system, the PC also tends to be more "user friendly" by offering graphical user interfaces (GUI) and simplified operating systems. The PC’s user interface is not as limited by past equipment decisions. Since mainframe equipment tends to be expensive, mainframe sites often continue to use out dated equipment because the life cycle and initial investment recovery time are long. The performance of the equipment may still be considered quite good. However, because of technology advancements and price declines, the capabilities the end user needs and expects out of the existing equipment are not there or are not supported. The PC is easier to manage and tailor to suit particular or specialized needs.

1.0 Introduction
Although the PC's capabilities are increasing, it still can not match the mainframe's centralized storage capability or processing power.
1.2 Existing PC/Mainframe 3270 Support

Up to the present time, the advantages of both systems have not fully complemented each other. This is especially true when the connecting network has a low throughput, which often means an asynchronous connection through a 3270 protocol converter. Systems, such as the IBM PC XT/370 and IBM 3270 PC, and terminal emulation boards [8, 10] have been marketed which better utilize the PC's increased performance within a 3270 environment. But these systems include either special peripherals or add-on boards that only support some aspects of cooperative processing. Cooperative processing by these machines included database query and extraction, file transfer, and limited virtual services and data [8]. In addition to the hardware expense, these systems and boards require a coaxial connection to the host which is often not available at remote sites. A protocol converter board may also be installed within the PC but requires a dedicated circuit to the mainframe's communication controller. One solution to improve 3270 access over asynchronous lines is to emulate the 3270 protocol converter within the PC's software by simplifying the mainframe's local protocol converter [3]. Although such an implementation can increase throughput of standard LU 2 sessions, improvements in the area of distributed processing would be hindered by unnecessarily burdening the PC with 3270 protocol conversion when its resources can better be applied elsewhere. Also, the end user may not be able to convince mainframe management to redefine the protocol converter ports.

A software solution which provides increased productivity in a 3270 environment has been presented by SIMWARE in a package called SIM3278. SIM3278 is a distributed software system that supports multiple host sessions, window management, file transfer, 3270 terminal emulation and on-line help [10]. Although SIM3278 is comprised of a remote and local

1.0 Introduction
module, it does not appear to support communications for other distributed applications. In addition, it is unclear whether this software package will work when connected through a protocol converter.

Other network connection methods exist which make use of SNA's LU 6.2 peer communication protocol. Unlike LU 2, which was designed to support display devices using the 3270 data stream, the LU 6.2 protocol was designed to support interprogram communication. LU 6.2 is not burdened with a data stream format but allows any data block to be transmitted independently of the communication protocol [12]. LU 6.2 is also referred to as "advanced program-to-program communication" (APPC) and indicates SNA's evolution into a distributed operating system, which is a key theme in the interaction between mainframes and workstations [9]. Unfortunately, many PC end users do not, at this time, have access to an LU 6.2 connection but are still working over LU 2 connections.

As a result of the physical connection, many PCs that are connected to mainframes act as dumb terminals which only provide file transfer and scripting capability. The ability to transfer files is very important because it allows mainframe output files to be used as input to PC programs which can, if necessary, reprocess and present data graphically. This methodology, although an improvement over terminal emulation, is not very interactive especially for iterative procedures. Examples of iterative procedures might include document preparation or program development. The mainframe's large storage and processing potential is not being fully utilized to help prepare information for graphical output; and no processing overlap is occurring.
1.3 Extending PC/Mainframe 3270 Support

The methodology of using transferred files as input to PC programs may be extended further by incorporating data transfer routines into application code that executes concurrently on the mainframe and PC. The data transfer routines can use the transparent mode of the protocol converter to send information to one another, like both sides of a file transfer program. Transparent mode disables the protocol conversions that normally occur and allows ASCII data to be transmitted from and to the host. The communication code could also be extended to provide a general transparent interface to support multiple applications. "Transparent interface," in this case, indicates that application data is independent or decoupled from the actual communication. The transmitted data is translated and framed for safe arrival at the receiving communication module which then removes the framing information and retranslates the data for the application program. This is a basic premise behind the ISO open system interconnection (OSI) reference model [1]. The reference model uses layered protocols to decouple applications from the specifics of the communication network.

Applying communication routines above the operating system is actually, in one sense, layering communication architectures to modify or hide the attributes of the underlying architecture. A communication architecture is a set of layered protocol modules that insulate the user or programs from the details of the communication network. In layered communication architectures the underlying architecture becomes the physical layer of the upper architecture which is seen by the application. In this case, communication routines are being applied over the existing 3270 protocol. The communication routines could consist of a complete OSI reference model to provide operating system type services, which are not normally present, to application programs. In general, the concept of

1.0 Introduction 7
layering communication architectures is not recommended because inefficiency might be added to the communication system. Since the protocol converters are in transparent mode, and thus the formatting introduced by LU 2 is removed, the PC requires only one active communication architecture. But the mainframe is experiencing the inefficiency of implementing an extra layer in preparing data for transmission.

The above discussion implies that a full definition of a communication architecture, as defined by the OSI reference model, could be applied over the default communication services. But as a first step, only a DLC layer was added to coordinate communication calls through the operating system to LU 2. By adding this extra layer, distributed programs can be supported over existing communication lines which employ a 3270 protocol converter. Through the introduction of concurrent processes the amount of communication necessary to execute a task, that normally operated using a file transfer methodology, may be reduced. The structure of a communication platform to support distributed applications, will be presented through the development of a SCRIPT WYSIWYG page previewer. An overview of SCRIPT previewing support for the PC will be presented next. Chapter 2 defines the DLC protocol, while Chapter 3 discusses how the page previewer can use the DLC layer entities for improved performance.
1.4 Distributed Page Previewer

SCRIPT/VS [28, 31] is an IBM text formatting program which uses embedded control words and tags within a source document. The source document is completely text based and further processing is required to view the final copy in a "what you see is what you get" (WYSIWYG) format. A WYSIWYG previewer displays a document on the computer screen exactly as it will be printed. Previewing support for SCRIPT documents is limited for PCs that gain access to the mainframe through terminal emulation. A text based preview may be accomplished from within the text editor by using the "DOIT" command. Since this method is text based, its capabilities are limited to previewing page and paragraph formats. Documents that incorporate graphics, special characters, or equations will be incompletely shown. As a result, PC users of SCRIPT often use hardcopy output while creating and editing a document. Such a document preparation cycle is inefficient in terms of time and resources. The major advantage to the "DOIT" command is that no special hardware or software is required.

IBM supports previewing SCRIPT documents over 3270 terminal connections. The "composed document viewing utility" (CDVU), which is part of the "graphics display and query facility" (GDQF), will preview SCRIPT and graphics data in a format "similar" to the printed output [17]. Graphical support for 3270 terminals is provided through the graphical data display manager (GDDM). GDDM provides application programs with an interface for sending bit-image graphics through the 3270 data stream. The receiving terminal must be equipped to interpret "program symbol set" (PSS) features prepared by GDDM. The use of GDQF, by a remote PC user, may be impeded by the significant transmission overhead generated through the use of the PSS. More importantly, terminal emulators do not
include PSS interpretation. GDQF is also a large package that may not be available at all sites which support SCRIPT.

A SCRIPT previewer, GDOIT, has been developed to specifically meet the need for page previewing capability on the remote PC [19]. GDOIT is a graphical "DOIT" command which is also invoked from within the mainframe editor. A terminal emulator script is used to monitor the processing and transfer of the SCRIPT file to the PC. A WYSIWYG preview is then produced through the emulation of an IBM 3812 page printer. An updated page previewer, demonstrating the communication platform, will be patterned after the 3812 emulator and will function similarly to GDOIT. The printer emulator will become a distributed application which uses the communication platform in place of a separate file transfer utility.

1.0 Introduction
CHAPTER 2
COMMUNICATION PLATFORM

The communication platform is a set of software modules that an application, which is distributed between a mainframe and PC, may use to handle, maintain and facilitate intermachine communication. The modules are designed to be completely independent of the application and yet are flexible enough to work with the application to reduce communication overhead. One of the final goals of the communication platform is to support interactive mainframe applications that utilize the PC’s hardware more completely. The platform should support applications that use the PC as a front end to the mainframe and provide a user friendly interface. Such applications may be communication intensive and so the communication platform should provide resources to improve performance over low throughput serial line connections.

To reach these goals the communication platform consists of a terminal emulator/command (TEC) module, a communication module, and a database module. The relationship of these modules to a distributed application on a PC/VM system is shown in Figure 2-1. The TEC module is responsible for initiating the PC to mainframe link and providing an environment in which to launch both the PC and mainframe applications. Note in Figure 2-1 that the TEC module is located only on the PC side since the system model considers the mainframe as a server and the PC as an initiator. The communication module is used by the launched applications for all intermachine communication. The database module’s responsibility is to manage primary and secondary memory resources to reduce communication overhead by providing temporary storage for iterative applications.
Figure 2-1 shows the communication platform running as PC and mainframe operating system applications. Figure 2-2 shows the communication platform as part of, or an extension of, the PC and mainframe operating systems. The communication platform could be considered a device driver to the PC or a nucleus extension to the virtual machine’s operating system. Applications would no longer link required communication platform routines but would use operating system calls for program communication. Eventually the mainframe and its resources could be considered as an extension of the PC’s operating system. The idea is similar to the virtual host storage [6] concept but is much more extensive. The host is not only viewed as a storage device but as a computing resource capable of running applications that interact with the PC’s resources. Such a configuration would require the user to know only one operating system. Refer to reference [8] for a discussion of distributed operating systems in a PC/mainframe environment.

Figure 2-1 shows the structure of the communication platform while Figure 2-2 indicates the possible evolution of the platform. The actual implementation of the communication platform for this thesis is shown in

2.0 Communication Platform
Figure 2-3. This chapter discusses each of the modules which make up the communication platform. The communication module will be discussed first because the scope of work involved the development of this module with an application. The insights gained through the development of the communication module can be employed in structuring the remaining modules and determining the support functions within them.

Figure 2-2: Communication Platform Evolution

Figure 2-3: Communication Platform Implementation
2.1 Communication Module

The communication module provides the distributed PC/mainframe applications with transparent I/O services. Applications that use transparent services do not handle any specific I/O details. As a result, the communication module should be able to handle or recover from transmission errors. It can accomplish this through the implementation of layered communication protocols. A communication protocol is a set of rules and heuristics that are followed by the sender and receiver to initiate and sustain information transfer in the presence of possible errors.

The communication module’s DLC protocol was modeled after the Kermit server file transfer protocol [4, 5]. The original Kermit protocol was developed at Columbia University to transfer files between a DEC-SYSTEM-20, IBM-370 series mainframes, and various other microcomputers. Although Kermit was used as the model for developing a protocol, the actual protocol is much more compact. The new protocol is concerned only with safely moving data packets between two machines whereas Kermit moves multiple files. Kermit was also designed to communicate with many types of computers and through various intermediate communication devices, so its protocol is very conservative and does not take advantage of any unique characteristics of the communication environment to increase throughput. The modular software design of the communication module attempts to overcome this "generality" limitation by allowing routines to be replaced or included as needed by a particular application. For example, the application can provide pointers to the compression algorithms the communication module should implement.

Figure 2-4 shows a block diagram of the environment in which the communication module was designed to work. The main performance limitations

2.0 Communication Platform
occur while communicating through the 7171 ASCII device attachment control unit. This particular device allows ASCII terminals, or personal computers running terminal emulator software, to emulate 3270 terminals. The 3270 protocol or data stream [36] is screen oriented and was originally designed to work solely with terminals. Consequently, no provision was made to accept large amounts of data from the terminal since all input was expected from a keyboard and no other input devices were anticipated. The terminals were character oriented and therefore only seven bit characters were needed. Because the protocol is display oriented, only screen size buffers were provided for I/O. But the 7171 and Series/1 protocol converters provide a transparent mode that allows screen size buffers of seven bit data to be passed from and to an application without the usual 3270 protocol conversions.

![Diagram](image)

**Figure 2-4:** Communication Environment

The 7171 supports two transparent modes, a transparent write and a transparent write/read. A transparent write allows the mainframe application to write a transparent buffer while all input from the PC is ignored. Only the attention identifier can be altered, by the PC sending an attention key sequence. Once the output data has been sent, the

2.0 Communication Platform
mainframe can then continue processing. Transparent write/read is similar, but input from the PC is allowed only after the complete transmission of a data buffer. The input from the PC is terminated by a carriage return character. Because the transparent modes operate in this manner, the data connection is effectively half duplex though the physical connection is full duplex. The 7171 is imposing limitations on how the serial link can be managed. This explains why Kermit is a stop-and-wait protocol and why sliding window protocols do not operate in IBM mainframe to PC environments (when connected via a 3270 protocol terminal server). For more information on the 7171 protocol converter refer to Appendix 'A'.

Throughput of binary data is severely degraded by only allowing seven bit data bytes to be transmitted and by the logical half duplex problem. The communication module's DLC protocol attempts to improve throughput by addressing the specifics of the communication equipment involved, namely the 7171. Improvements are also made through compression and by working closely with the application to allow processing overlap.
2.1.1 DLC PROTOCOL DESCRIPTION

This discussion refers to the mainframe as a host, server, or remote system. The PC may also be referred to as the local system. The packet protocol does not function until the local and remote communication modules have been initialized. Currently this is the responsibility of the application. The following steps currently define how the communication modules begin functioning.

1) The user informs the terminal emulator/command (TEC) module to start an application.
2) The TEC module informs the remote operating system to start the application’s remote code.
3) The remote application then uses the linked communication module routines to initialize the communication path and send a message to the TEC module to start the local application. At this point the remote application is communicating in transparent mode and is awaiting input.
4) The TEC module starts the PC side of the application.
5) After the local application initializes, the communication link is fully functional with the mainframe being in a server-command-wait state. Packet communication may now proceed.

Once communication is established, the PC is always considered the initiator. The PC sends orders and the mainframe fills the order, normally by sending data packets. Once the order has been filled, the mainframe sends a ready packet to inform the PC communication module of an order completion. An order is always application dependent. That is, an order has no meaning to the local or remote communication modules which view the order as just another data packet to be passed to the application. Therefore, no order packet is defined within the communication protocol. The concept of orders conforms to the half duplex
nature of using transparent mode. The order terminology also reflects calling the remote system a server.

Note that there are actually two protocols in effect. The application must enforce its own communication strategy, i.e., the data within each packet must be meaningful to the receiving application. The communication module’s protocol is only concerned with passing that data to the application. So the idea of orders actually belongs to the application’s communication strategy (communication strategies are discussed in Chapter 3). The ready packet is significant to the local communication module and must be initiated by the application when an order is filled or to accomplish line turn around. Having the application issue ready packets violates keeping the two protocols strictly separate and was allowed simply for performance reasons, but does not necessarily need to be implemented. The protocol implementation details will demonstrate the reasons behind this decision.

Tables 2-1 and 2-2 show the DLC packet types and the response of the local and remote systems upon packet reception. A state table is not needed to describe the communication module’s protocol because the next state, after a reply, is always a receive state. Although a state table may be useful when describing an application’s communication strategy. Table 2-2 indicates that the mainframe does not implement a timeout or retry procedure. This follows as a direct consequence of using transparent mode and having the logical half duplex problem. Implementing a timeout on the mainframe would be costly because of the extra processing time required. A timeout function is only needed on one system. The PC, as the initiator, is the best choice. Note that an abort packet is only defined for the remote system. The PC’s communication module uses the abort packet when an unresolvable error is encountered. The appli-
cation may also instruct the communication system to send this packet in the event that it also encounters an unresolvable error. An application's communication strategy should incorporate an exit procedure for terminating orders and shutting the application down gracefully.

<table>
<thead>
<tr>
<th>Table 2-1: Local System Packet Reception</th>
</tr>
</thead>
<tbody>
<tr>
<td>Received Packet</td>
</tr>
<tr>
<td>-----------------</td>
</tr>
<tr>
<td>Data</td>
</tr>
<tr>
<td>Ready</td>
</tr>
<tr>
<td>ACK</td>
</tr>
<tr>
<td>NACK</td>
</tr>
<tr>
<td>Bad Data</td>
</tr>
<tr>
<td>Time out</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 2-2: Remote System Packet Reception</th>
</tr>
</thead>
<tbody>
<tr>
<td>Received Packet</td>
</tr>
<tr>
<td>-----------------</td>
</tr>
<tr>
<td>Data (n)</td>
</tr>
<tr>
<td>Data (n-1)</td>
</tr>
<tr>
<td>ACK</td>
</tr>
<tr>
<td>ACK (n-1)</td>
</tr>
<tr>
<td>Abort</td>
</tr>
<tr>
<td>Bad Data</td>
</tr>
</tbody>
</table>

The communication protocol should follow some general guidelines:

1) The local communication module should always resend the last packet in the event that bad data was received or a timeout occurred. This explains the absence of a NACK packet definition in Table 2-2 and avoids the endless cycle of sending negative acknowledgments of received NACK packets.

2.0 Communication Platform
2) Duplicate packets should be discarded and the proper acknowledgement sent. In practice, duplicate packets should only occur on the remote side because of heuristic one. Remember that only the local side has the ability to time out.

3) If the local communication module times out or receives a NACK after sending an ACK packet then the local module must send a sequence number with the acknowledgement to avoid losing or skipping a packet. This heuristic handles error situations arising from automatically acknowledging data packets. Automatic acknowledgement is explained in the protocol implementation section.

The packet format is shown in Figure 2-5. Data is framed by a packet header and tail which are used to verify data integrity. Since eight bit character transmission can not be guaranteed, all the fields of the header and tail are always defined to be seven bit fields. The header and tail are never prefixed and so many of these fields are coded into the printable ASCII range (20h - 7Eh). Prefixing is described below. Translation to the printable ASCII range depends on the character restrictions imposed by the communication link.

<table>
<thead>
<tr>
<th>Packet Header</th>
<th>Packet</th>
<th>Packet Tail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mark</td>
<td>Length</td>
<td>Format</td>
</tr>
</tbody>
</table>

**Figure 2-5: Packet Format**
The packet fields are defined as follows:

**Packet Header:**

*Mark Character* - A synchronization character indicating the start of a packet. The mark character is never prefixed or translated to printable ASCII.

*Length* - The packet length includes the remainder of the header and extends up to, but does not include, the packet tail. Two bytes define the packet length and another two bytes represent the bitwise complement of the packet length. The complemented bytes are an attempt to ensure the data integrity of the length field. The maximum packet length is \(2^{14} - 1 = 16383\) bytes. If printable ASCII translation is required then the maximum packet length is \(2^{12} - 1 = 4095\) bytes.

*Format* - The format byte, as shown in Figure 2-6, specifies the packet type as well as the process used to format the packet data. The following list summarizes the packet types and associated codes that are recognized by the communication module:

- **2xh** - Data packet
- **30h** - Acknowledge (ACK) packet
- **40h** - Negative Acknowledge (NACK) packet
- **50h** - Ready packet
- **60h** - Abort packet

The codes for each packet type fall within the printable ASCII range (20h - 7Eh). The low nibble of the format byte is used by the receiving communication module to decode the packet. Bit positions zero and one are used as an index to the application supplied decompression routines. An index of zero indicates that the data is uncompressed. A zero in bit position two signifies that 7 bit to 8 bit conversion is disabled. A zero in bit position three indicates that prefixing is disabled.
Sequence - The packet sender numbers each packet sequentially so that the receiver can detect lost packets, and if necessary the receiver can inform the sender which packet was lost. Sequence numbers are coded into the printable ASCII range.

Packet Data:
Compression is applied to the application's data. The actual amount of data sent will depend upon the compression ratio as well as the intervening communication equipment. The amount of data will not exceed $2^{14} - 3$ (16k - 3) bytes as defined by the packet length field.

Packet Tail:
Checksum - A fourteen bit arithmetic checksum is computed on all the bytes from, but not including, the mark character up through the data field. If printable ASCII translation is required then only 12 bits of the checksum are employed.
EOP - The end of packet is not necessary on all systems. It is mainly for the benefit of the intervening communication equipment. The receiving communication module ignores this byte if it happens to survive the trip to the receiver. The EOP character and mark
character must not be identical. The EOP character is never pre-
fixed or translated to printable ASCII.

The length of the packet header, tail and data may vary depending upon
the direction the packet is moving. This is a direct result of the
intervening communication equipment and by an effort to optimize perfor-
mance. Limitations which affect packet length fall into three catego-
ries: data character size, removal or interpretation of data characters
within the packet stream, and limited buffer length. The first two
limitations generally serve to degrade throughput performance while
sending binary files, whereas buffer length limitations may degrade per-
formance depending upon the data channel noise. Shorter packets are
preferred on a noisy channel so that the cost of retransmission is
lower. The 7171 imposes limitations in each of these areas. Refer to
Appendix 'A' for a description of the 7171 and the limitations it impos-
es on communication.

The communication module addresses character size limitations by employ-
ing an eight to seven bit conversion routine to the packet's data field.
The packet header and tail are always assumed to be seven bit data. The
conversion routine adds a constant 12.5 percent overhead to the data
length. The application may inform the communication module to disable
conversion in the event that ASCII data is being transmitted or the
compression routine generates variable bit length codes. Kermit handles
transmission of eight bit data over seven bit data lines by either
controlling the parity bit or by using eighth-bit prefixing. Eighth-bit
prefixing adds an extra character into the data stream to prefix each
byte that has its most significant bit set. Eighth-bit prefixing of a
binary data stream will probably add significantly more than 12.5 per-
cent overhead to the data field.

2.0 Communication Platform
Character prefixing is used to notify the receiver that the next character has been coded in some predetermined fashion to prevent intervening communication equipment from interfering with the packet. Kermit translates all non-printable ASCII characters (0 - 31, 127) to the printable ASCII range (32 - 126). Each translation is prefixed by a specific printable ASCII character. If the prefix character is in the data stream then it also is prefixed. For a binary data stream with a uniform probability distribution, 12 percent of the data stream is overhead. The communication module’s protocol only prefixes characters which disrupt packet communication, along with the prefix character. Assuming a uniform probability distribution, the transmission overhead is usually less than two percent. The communication module allows prefixing to be disabled while transmitting in one direction or the other. For example, no prefixing is required while transmitting to the PC but the opposite is not true.

The communication protocol only implements data compression on the packet’s data field. Compression occurs before the packet is framed with the header and tail information and before any prefixing or eight to seven bit conversion. This ensures that the packet is properly delimited and includes correct prefix characters. Additionally, each packet will contain close to the maximum packet length available which will reduce transmission overhead. Having compression occur before packet formation also allows the interrupt handler for the PC to process incoming characters during a single interrupt while leaving the application ample processing time. Decompression during an interrupt would not allow background communication to occur without noticeably slowing down the application. The communication module implements the compression and decompression routines through application provided function pointers. The application has the flexibility of choosing compression algo-
algorithms which best suit the data being transmitted. The application may also disable or change the algorithms dynamically to suit different data types.
2.1.2 PROTOCOL IMPLEMENTATION

A functional block diagram of the input and output packet handlers of a communication module is shown in Figure 2-7. The local and remote communication modules exhibit the functionality shown in Figure 2-7 but are operationally different. The differences result from a processing overlap, the communication hardware for each machine, and the protocol's definition of a server. These operational exceptions prevent much, but not all, of the code from being interchangeable between the local and remote communication modules, although the interface to the application on both sides of the communication link is similar. Because of such differences, the PC implementation of the protocol will be covered separately from the remote module's implementation.
Figure 2-7: Communication Module
Local System Protocol Implementation

The primary goal of the protocol implementation, in the local communication module, was to allow background communication to occur. The mechanism for background communication on the mainframe is already in place as a result of its virtual machine environment. In terms of each machine processing power an I/O processing overlap is needed to a greater extent on the PC. The time required by the PC to wait for and process I/O is not compensated for by an overwhelming processing speed such as the mainframe possesses. The PC is also hampered by the necessity of a user interface. The processing power and overhead required for a graphical user interface can bog down many PCs. As a result, the input packet handler is designed as a two level finite state machine. Applications that implement event driven user interfaces can easily establish the communication module into the message passing or polling loop.

Input arriving at the PC is handled by an interrupt service routine (ISR). The "decode packet" block of Figure 2–7 represents the functionality of the ISR. The ISR checks the incoming packet's checksum and places the data count, packet type, sequence number, and data into a circular buffer for the packet data input routines. A packet format error may cause a packet to be ignored which will result in an eventual time out recovery. A checksum error will cause the ISR to place a flagged sequence number in the circular buffer so that the protocol arbitrator can issue an appropriate response. The ISR processes an entire packet in nine states. A concern of the ISR design was that the longest state should consume no more than 33 percent of the available time between the reception of successive characters. The 33 percent limit is a first estimate of the percentage of processing time that may be taken away from the application without noticeably slowing down the
application. This translates into a design constraint of 825 CPU cycles for a 4.77 MHz PC receiving characters at 19,200 baud using 10 bits/character (1 start bit, 7 data bits, 1 parity bit, 1 stop bit).

To increase throughput the ISR will automatically acknowledge data packets whose checksum is correct. This avoids the delay of having the packet data input routines process almost all the data within the circular buffer before a response can be generated by the protocol arbitrator. A ready packet will not generate an automatic acknowledgement and so orders, terminated by ready packets, can be filled without any intermediate intervention by the application. If the communication module was not allowed to examine ready packets then the application’s protocol would have to supply order continuation responses, or new orders, after the reception of each packet. This situation would be inefficient but would greatly simplify the low level protocol since a response would only have to be generated by out of sequence packets, bad packets, or in the event of a timeout error. Additionally, each application would be required to develop a more complex protocol.

The packet data input routines fetch data from a circular buffer and are designed to work as the second level finite state machine. The input routines work to return a string of decompressed data on each call by the application. When a new packet is received the protocol arbitrator checks the ISR’s response and the sequence number and generates any necessary responses. If the sequence number is correct then the packet type is decoded to activate prefix removal, 7 to 8 bit conversion, and decompression as necessary. Data is passed sequentially to each routine, in the above order, until a return string is formed or until the packet is depleted. The 7 to 8 bit conversion routine and some decompression algorithms can have data remaining in their local buffers.
following an order completion. A ready packet informs the communication module to flush the buffers of these routines so that any remaining data is returned to the application.

The communication module's output routine is character oriented. The characters are compressed, converted from 8 to 7 bits, prefixed, and then stored in the packet data buffer. The routines are implemented in the above order. Any or all of the routines may be disabled. When the packet data buffer is full or the application issues a flush order then a buffer pointer, character count, and packet type are passed to the form packet entry routine. The packet type is coded according to the enabled conversion routines at the time of the flush. The form packet routines frame the data with the header and tail information. A copy of the packet is saved in an auxiliary buffer in case the protocol arbitrator requests a resend of a packet. The packet is then sent to the remote communication module by the transmit buffer routine.
Remote System Protocol Implementation

The remote communication module is functionally similar to the local system's module, as diagrammed in Figure 2-7, but with three operational exceptions. As noted above, these exceptions result from overlapped I/O and processing, differences in communication hardware, and from the protocol definition. The description of the remote system's communication module will concentrate on these differences.

The remote communication module does not take advantage of the mainframe's inherent ability to overlap I/O and processing because of the increased code complexity. The absence of such an overlap should not pose any significant degradation in performance for most applications. Remote applications whose computation time is large when compared to the communication time would benefit the most. Further development of the remote communication module should probably incorporate background communication. Implementation of such an I/O structure would cause the remote module to be more operationally consistent with the local module. Since no processing overlap is present, the remote communication module lacks an ISR and remote output is directly followed by input. Therefore, the remote module's output and input routines are coded within a different framework from the local module. So the separation of each function, is not as well defined as Figure 2-7 would imply for the remote communication module.

The mainframe's communication hardware is buffer oriented as opposed to the character orientation of the PC's UART (the use of a FIFO buffer on some UARTs is an exception). Packet routines also reflect whether the communication equipment is buffer or character oriented. As a result, the remote module's packet decode functions operate on whole packets.

2.0 Communication Platform
This contrasts the local module's ISR which processes one character per interrupt. An ISR on the mainframe would operate on an entire packet for each interrupt. The mainframe's communication hardware also requires that an introducer sequence be attached to the beginning of each output packet.

The remote and local protocol arbitrators respond to inputs differently. The response varies, according to Tables 2-1 and 2-2, because the remote module is considered the server whereas the local module is the initiator. As a result, the code for the arbitrators is not interchangeable.
2.2 Terminal Emulator/Command Module

The terminal emulator/command (TEC) module is used to initiate the PC to mainframe link and provides an environment in which communication platform applications may commence execution. To accomplish these tasks the TEC module must be capable of terminal emulation, automatic execution and interpretation of predetermined key sequences through a scripting language, and the ability to execute external programs. Presently, these features are provided through the use of a commercial communication package called HyperACCESS/5 [11].

Terminal emulation is required within the 3270 environment for log on, interaction with the mainframe's operating system, and for running programs that do not use transparent I/O. The 7171 translates the 3270 data stream into the escape codes of the terminal being emulated on the PC. Continuously operating the 7171 in transparent mode by trapping and filtering mainframe output might eliminate, except for logon, the need for PC terminal emulation. Whether such a proposal is practical (or even possible within a user's virtual machine) depends on whether communication performance of full screen 3270 applications could be improved. Application of this concept could be expanded to unify the PC and mainframe operating systems or hide the details of mainframe operating system commands. The TEC module could then serve the same purpose to both operating systems as MS-Windows does for DOS. Such a TEC evolution would no longer require the use of a scripting language.

A scripting language is used to control the execution of multiple programs that form a complete application. A script is initiated by the user pressing a special key sequence or "hot" key to start an application such as the page previewer. A script can issue commands to the mainframe to start programs or execute operating system commands.

2.0 Communication Platform
script may then interpret the resulting output and make decisions concerning the continuation of the application. In the case of the page previewer, a script sends an editor file to the document composition facility and, by monitoring the mainframe’s responses, accepts a PMP file from the user’s reader. In addition, the script starts the remote and local sections of the page previewer interpreter and then returns control to the editor after the page previewer exits. The linking together of these programs and the interpretation of the resulting output could be accomplished without the use of a scripting language, but this feature of the communication package simplifies development. In either case the communication package must be able to execute external programs, on the PC, without affecting the communication line. The local side of an application must be able to receive and return control to the communication package without dropping the communication line.

The present TEC module configuration, which uses HyperACCESS/5, has some definite limitations. The communication package tends to be large because it supports many features, such as file transfer, that are not required. Integration of the communication platform modules and the control of memory usage is hindered by the lack of source code. The amount of memory the communication package releases to an application may not be sufficient. As a result, the page previewer can only review small documents (approximately 30 to 40 kilobytes of data). The cost and availability of the communication package can also hinder the wide spread use and development of applications. Other commercial and shareware communication packages exist that contain scripting languages [2]. Scripting is a recent addition to Kermit [7] and the use of Kermit is very attractive since it is freely distributed and the source code is readily available.
2.3 Database Module

The database module is envisioned as a memory manager for large iterative applications that contain subprograms. An example of a large iterative application would be a word processor that consists of an integrated editor and page previewer. The database module could be used by subprograms to store data over multiple invocations for the sole purpose of reducing communication.

The page previewer could take advantage of such a capability for storing font patterns which can be a large percentage of document data. During a typical editing and previewing process, the font patterns and many pages will not change dramatically. If only the edited pages, and any additional patterns, are retransmitted then the communication overhead should decrease substantially. Proceeding a step further, many font sets could be permanently located in local secondary storage so that font patterns would not require transmission. The remote application may not even need to transmit a fonts list depending upon how SCRIPT numbers and names fonts. The database module could then take on the role of a cache by storing in memory the most recently and frequently used font patterns.

Each new application could implement its own code to accomplish this type of task. But having a system database module, that the TEC can control, solves the problem of cleaning up memory or disk storage in the event that a subprogram encounters an unresolvable error which terminates the application. A separate database module would also free the application programmer from having to write system dependent I/O code. The database module could concentrate on providing flexible and fast storage that varies depending upon the hardware available.

2.0 Communication Platform
CHAPTER 3
PAGE PREVIEWER APPLICATION

The use of the communication module is demonstrated by the implementation of a SCRIPT WYSIWYG page previewer. This particular application does not require the full definition of the communication module as described in Chapter 2. However, it does serve to establish the concepts and software design decisions that are required for an application to make use of the communication module. The software design decisions depend upon the application and fall into two general categories:

1) The division of the application’s algorithms between the PC and mainframe so that communication is minimized to increase performance.

2) The communication strategy which involves the content and interpretation of packet data, data compression algorithms, and the type of interface between the application and communication module.

This chapter presents an overview of the SCRIPT page previewer followed by discussions of application division and communication strategy.
3.1 Page Previewer Description

SCRIPT is a text formatting program which is also referred to as the document composition facility (DCF). SCRIPT accepts text files with embedded control words and tags that govern the printing characteristics of each part of a document. The control words and tags allow various fonts, including mathematical, and formatting controls, such as centering, to be included in an ASCII or EBCDIC text file without the use of control characters. Besides the inclusion of a mathematical formula formatter, SCRIPT also allows references for the insertion of graphics figures. Because the source file is completely text based, further processing must be provided to view the document as it will actually appear when it is printed.

DCF produces a collection of device independent printing instructions and data known as objects. The output objects from DCF contain textual information, such as fonts and character placement data, along with vector information for the support of boxes and lines. Commands which indicate the placement of more extensive vector or bit mapped graphics are also included with references to external objects. SCRIPT, or DCF, is one type of object producing program. Other object producing programs include "graphical data display manager" (GDDM), "graphical display and query facility" (GDQF), overlay generation language, print management facility, and the print services access facility. A complete document may need objects derived from multiple programs.

All of the objects for a single document are assembled by another program, which resolves all external object references and produces a device dependent data stream for a particular series of printers. The "print service facility" (PSF) serves the IBM 3800 printer while the "composed document printing facility" (CDPF) handles the IBM 4250 print-

3.0 Page Previewer
er. The 3812 page printer, in a VM environment, is serviced by VM3812. These programs, or print facilities, support what IBM refers to as advance function printers. Advance function printing is the capability of printing pages that contain a combination of different fonts, bit image graphics, and vector graphics. Advance function printing is often referred to as page mode printing since the printer builds a complete page image in memory before the actual printing takes place. Figure 3-1 summarizes the document printing process. The page previewer, which will demonstrate the communication platform, emulates the IBM 3812 advance function printer to produce WYSIWYG output.

![Diagram of Document Printing Process]

**Figure 3-1:** Document Printing Process

3.0 Page Previewer
The 3812 printer emulator was developed to work as a functional extension of the mainframe editor, which is usually XEDIT in VM. A preview was accomplished by entering a "hot" key sequence while in XEDIT. This would inform the communication software to invoke a script program. The script program would in turn send the XEDIT file to DCF, via the 'DOIT' command. DCF produces an object file that is automatically sent to the print facility, VM3812. The options used by 'DOIT' cause DCF and VM3812 to produce an output file that does not contain 3812 printer query or dialogue commands and is redirected to the user's virtual machine.

Query and dialogue commands are only used when VM3812 is producing hard copy output to a physically attached printer. The output file is a series of page map primitive (PMP) commands. When the script program detects the arrival of the PMP file, it transmits the file to the PC via Kermit. After the complete file has been transferred to the PC, the printer emulator, GDOIT (graphical DOIT), processes and displays the document. When previewing is complete, the script program returns the user to XEDIT. For more information and a description of GDOIT refer to reference [19].

The previewing sequence remains as previously described. However, the printer emulator is a distributed application which uses the communication modules in place of Kermit. The previewing process is shown in Figure 3-2. The printer emulator application was completely rewritten in a more modular form to ease the transition to a distributed application and to incorporate a number of changes to the user interface and PMP interpreter. The user interface has been coded in an event-driven style to allow background communication and the use of a mouse. Larger viewing windows were added that accommodate scaling, panning, and rotating of document pages. The revised PMP interpreter includes new data structures and algorithms for faster pattern display and support of all

3.0 Page Previewer
the possible PMP orientation commands. Improved error support was added and a relative decrease in code size was obtained.

Figure 3-2: Document Previewing Process
3.2 Application Division

The goal of dividing up an application program is to improve the application's performance and capabilities. If performance is to be improved, the present application's bottlenecks and any inherent parallelism must be identified, and the methods the application uses to accomplish its tasks must be understood. Most applications differ from one another and division is not necessarily the remedy for a sluggish program. Changing and modifying algorithms and data structures may also be required. The very process of distributing an application and using the available hardware effectively will likely require algorithm and data structure modifications. Application division depends not only upon the application's function, but also upon the machines and network over which the program will execute. The strengths and capabilities of each machine should be recognized and combined to overcome application and system bottlenecks, such as communication.

The time required to transfer the PMP data file to the PC, using Kermit, is a major bottleneck within the GDOIT previewer. The communication module replaces Kermit and should, for the same amount of data, improve throughput for the new previewer. But further improvements, to minimize communication, may be devised by considering what data should be transmitted and the form of that data. The GDOIT previewer reprocesses the PMP file on the PC to extract the page commands and pattern data while removing unnecessary or unsupported commands and file formatting information. The whole document is reprocessed by GDOIT before any pages are displayed. This processing step could be quickly accomplished on the mainframe. The remote code of the new page previewer reduces the communication load by, on average, more than 5 percent through preprocessing the PMP file on the mainframe.
GDOIT interprets the PMP commands to build a screen image of each page. The question arises as to the efficiency of moving the PMP interpreter to the mainframe and then transmitting bit images to the PC. The resolution of the IBM 3812 page printer is 240 pels/ inch. To store or build a complete bit image of an 8.5" X 11" page would consume 657.4 kilobytes. Typically, two to four megabytes of virtual storage is available to each user on VM, which is adequate to store six document pages along with the PMP interpreter and the communication module. Transferring the complete image to the PC is impractical, since the PC’s limited memory is unable to store one page and the transfer time, even at 19,200 baud, would be unacceptable. Even with a superior compression routine, the time necessary to transfer just one 657 kilobyte page would be much greater than the overhead of transferring and interpreting a multipage PMP file. A large, feature packed PMP page is about 15 kilobytes whereas a "typical" PMP page is approximately 3 kilobytes. The amount of storage required for pattern data is difficult to access on a page basis due to the fact that a pattern may be used on many pages and would only be transmitted once.

Another division method would be to transfer only the bit images that the PC requires. A full screen window (640 x 440) consumes 34.4 kilobytes of memory. Again, the time necessary to transmit the bit image, even when compressed, is much greater than transmitting the PMP commands and running the interpreter to display the page. In addition, the PMP data is transmitted once whereas another image transfer would have to occur every time the user selected a different viewing area or changed scales. As a result, a PMP interpreter is resident on the PC and PMP data, not bit maps, are transferred using the communication module.

3.0 Page Previewer
Since the PMP interpreter and output routines are resident on the PC, they were rewritten with new algorithms and data structures that increased page display performance. A description of the pattern display algorithm may be found in Appendix ‘C’. The actual time necessary for PMP interpretation and display varies depending upon the page content, the current window size and scale, as well as the machine class. Additional performance increases may be accomplished by selectively coding some or all of the PMP interpreter code, and especially the output routines, in assembly language. Screen output consumes a large percentage of time compared to the entire output algorithm. The low level monitor routines, accessed from the C language, add an extra layer of software which could be replaced by specially written screen drivers to increase performance.

The division of the page previewer may be characterized by the location of general code modules. The mainframe houses a PMP file preprocessor which separates page and pattern data and then places the data in memory. The file preprocessor also flags or discards unsupported PMP commands (refer to Appendix ‘B’ for a list of PMP commands which are supported by the interpreter). A remote communication module is also resident on the mainframe to replace Kermit. The remote application requires an additional module, referred to as a page/pattern manager, to interface with the communication module and fetch data required by the PC. The PC contains a local communication module and the PMP interpreter which are linked together by a user interface. The communication strategy discussion details how these generalized code modules fit together to form the complete application.
3.3 Communication Strategy

Each application that uses the communication platform must develop its own communication strategy. The communication strategy comprises the software decisions that the application employs in using the communication platform. The decisions generally involve the content and interpretation of packet data, the use of compression algorithms, and the interfacing of the application with the communication module. The communication strategy indicates how an application is using the communication platform and more specifically, in the present version, how the application is using the communication module. The communication module is only concerned with passing data between the two sides of the application and is not concerned about the content of the data. As a result, the data which is passed must be meaningful to the receiver in order that both sides of the application cooperate to accomplish the overall task. In addition, the data should be formulated to minimize communication and transmitted so as to minimize the apparent communication overhead. Communication strategies will vary depending upon the application. Guidelines for developing a communication strategy for general applications will be outlined followed by more specific information which is applicable to the page previewer.
3.3.1 Development Guidelines

The communication module, because of its restricted environment, imposes some rules on the application to enforce line synchronization and to avoid time out errors. The distributed application must communicate in an orderly manner by alternating talking and listening. The present version of the communication module does not stack output requests for transmission at the next line turn around. An application may implement this type of order handler as part of the interface to the communication code. To maintain line synchronization, the communication module was designed with the intent that the PC initiate orders while the mainframe acts as a server. Therefore, when the PC initiates an order, the remote side of the application should always respond to the received order or data. Also, the remote side should always end an order completion with a ready packet to prevent automatic acknowledgement, flush communication buffers, and return control to the PC.

In developing a communication strategy, the content or type of data must first be determined. Distributing the program partially accomplished this step. It was determined that transmitting PMP data is more efficient than transmitting bit map images. Another aspect of determining the data content involves logical compression. Logical compression involves representing data such that its original form can be reformed or recalculated. For instance, the date '1 January 1992' could be stored as is, or in the form '010192'. Logical compression does not take advantage of redundancy or the frequency of character occurrences. Logical compression is solely application dependent [20]. The implementation of logical compression will depend upon the interrelationship of the data items as well as the amount of extra processing that can be tolerated.
Next, the application designer should consider the organization or format of the data transfer. Should the data be transmitted as a single block, similar to a complete file transfer, or in discrete chunks as the receiving station requests the data? The answer depends upon the amount of data and the complexity of the application code. If the amount of data exceeds the PC's memory capability, then the PC code will require the capability to request particular data items or have access to secondary storage for temporarily saving data. The ability of the receiver to request particular data items indicates that the sender must be able to organize the data set for quick retrieval, which increases code complexity.

The type of, and the response of the user interface will also help determine the data transmission format. If a highly responsive user interface is required, background communication and the ability to request particular data items may be necessary. These capabilities require code with increased complexity on both sides of the application. The user interface must apportion a time slice to the communication module which will depend on the data present in the local machine as well as the data the user is requesting. The application code should take advantage of the finite state nature of the local communication module. Additional code may also be necessary on the local machine for keeping the communication line busy in order that the apparent communication overhead is decreased. The remote machine's code may need more "brains" to avoid sending redundant data. Some data may have already been transmitted that partially fulfills the present request.

The amount of data and the desired response of the user interface help determine the data transmission format and the code necessary to interface with the communication module's code. Once the data transmission
format has been determined, physical compression schemes can be analyzed for their effectiveness in compacting data as well as determining the overhead required for their implementation. Physical compression will be considered in a separate section because of its importance and relative effectiveness in reducing communication overhead. The steps in determining a communication strategy may be summarized as follows:

1) Determine the data type which conveys the necessary information while reducing transmission time.
2) Apply logical compression where applicable.
3) Identify the most useful data transmission format in terms of the user interface, code complexity, and data quantity.
4) Analyze and implement physical compression techniques.

The individual steps, as outlined above, may be obvious for some applications. The level or degree to which a step may be implemented might be done gradually to reduce the complexity of debugging a distributed application. For instance, a new application might forgo logical and physical compression while the user interface operates sequentially to avoid background communication. Compression and operational features could be added later to improve performance.
3.3.2 Page Previewer Organization

The steps outlined in the previous section can be applied to the page previewer application. As previously mentioned, distributing the application indicated that the page data should be transmitted in the PMP format. PMP page commands could actually be considered a form of logical compression applied to a document page. As a result, no further logical compression was applied to the PMP page commands. The PMP pattern data consists of a header with relative position information followed by the bit image of the pattern. The bit image is not readily subject to logical compression, however, it could be physically compressed prior to inserting the pattern into the data stream. Although not undertaken, developing a relative encoding [20] scheme for the bit mapped patterns may be another avenue for decreasing transmission time.

An efficient data transmission format is based on sending individual pages and individual patterns as requested. To accomplish this, the remote page/pattern manager must be able to separate and associate individual pages and patterns. Higher level orders may then build on fetching a pattern or page, such as a "send document" command, and are easily implemented. Separation and retrieval of individual pages and patterns also improves the handling of large documents in terms of local memory management and in selectively displaying pages quickly without transmitting unneeded pages and patterns. Initial development of the distributed page previewer is based on sending the entire file because the remote page/pattern manager can only selectively find and transmit individual patterns and pages; it cannot yet associate which patterns belong to a specific page to avoid sending redundant information. The page previewer is an application that requires considerable user interaction and so an efficient user interface is desirable. Selective
transmission of pages with their associated patterns allows faster page display and the ability to download, in the background, page data within the vicinity of the page currently being viewed.

Figure 3-3 shows the organizational structure of the page previewer. The user interface for the page previewer consists of the command handler coupled with the event polling loop. Inputs from the user originate from the keyboard or mouse. The order manager is also polled and indicates whether an order is pending and indirectly the status of the communication module. If a poll operation indicates that no orders are pending, the command handler will issue a new order for the remote application to transmit another page. An algorithm may be employed that attempts to "guess" the next most likely page to be viewed. If the order manager is polled and an order is pending, the input data manager is given a time slice to organize and store some of the incoming data. The input data manager returns control to the order manager with an indication of order completion. The page/pattern manager, after transmitting all the data to fill an order, sends a ready signal to the input data manager to indicate order completion.

An example will clarify the previewer organizational functions. When the page previewer is invoked, the command handler sends out an order for a file header that was prepared by the PMP file processor. The header indicates the number of pages in the document and whether errors were encountered while processing the file. If no preprocessing errors were detected, the command handler sends out an order for page one, initializes the user interface, displays a downloading indicator, and then enters the event polling loop. At this point the user may select a page for previewing, such as page ten. The command handler intercepts the keyboard or mouse input and calls the display page code, which in

3.0 Page Previewer
Figure 3-3: Page Previewer Organization (Local side)

Figure 3-3: Page Previewer Organization (Remote side)

3.0 Page Previewer
turn finds that page ten is not in local memory. So the page display routine waits for the current pending order to be filled and then issues an order for page ten. When the page order is filled, page ten is displayed and control returns to the polling loop. The polling loop indicates that no orders are currently pending and so the command handler issues an order for page eleven, which is transferred while the user views page ten.

The page previewer is an application in which data transfer is largely unidirectional towards the PC. The PC also issues orders which are only a few bytes long. As a result, the full definition of the communication module was not implemented. The local output and remote input routines do not use packets or compression so that memory and processing time are conserved. Such a savings is especially helpful for the PC. In this case the application has direct access to the transmit buffer routines located in the communication module.
3.4 Physical Compression

An application's communication strategy also involves the use of compression. The communication module does not contain a default compression algorithm since the effectiveness of compression routines depends upon the type of data being transmitted. Simple algorithms, such as run length encoding, may not yield enough compression to justify the code overhead, whereas complex algorithms may be too slow. Three lossless compression techniques were examined for the page previewer's data: run length, Huffman, and Lempel-Ziv-Welch (LZW). Each of these techniques has advantages and disadvantages in terms of algorithm complexity, quickness, memory consumption, and compression ratios. Because of the page previewer's WYSIWYG function, no lossy algorithms were considered to improve compression ratios. In addition, compression was examined only at the communication level and not within the application. An application may further reduce communication overhead by examining and applying compression to each individual data type before bringing all the data together into the communication stream. For example, the page previewer's font patterns are bit mapped and may respond favorably to a relative encoding technique.

Table 3-1 summarizes the results of compressing PMP document files for the compression algorithms tested. The PC compression algorithms PKARC and PKZIP are only included in Table 3-1 as a reference point for those familiar with their performance. PKARC and PKZIP were not considered for inclusion into an application because they do not operate on asynchronous data streams. The files named "slide" and "schedule" are one page documents used as examples in Appendix 'D'. The files "slide5" and "slide10" are the same document as "slide" with the exception that the document has been repeated on five and ten pages respectively. The files "turing", "turbtest", and "resume" are multipage documents which
are too extensive for inclusion into this report, but these files are well characterized in Table 3-1 by the page and pattern data. Compression results were calculated for the packet data that occurs when the local application transmits a "send document" order with eight to seven bit translation disabled. The results are shown in Table 3-1 under the label "File Compression". Similarly, the results for sending the pages and pattern data individually are also shown in Table 3-1. Each compression algorithm allows compression to be extended over multiple packets.

The results from Table 3-1 indicate that LZW compression is more effective than the other compression algorithms tested. LZW compression is both computationally expensive and memory intensive, and would therefore be unsuitable for transferring data to a remote application when the local application requires an efficient user interface. Since the page previewer's data is coming from the mainframe, LZW compression was implemented into the page previewer's communication strategy. The buffer space required for LZW decompression on the PC is extensive but justifiable for the reduction in communication. The following description of the tested compression algorithms, along with their characteristics, should clarify how an application may determine an appropriate compression algorithm.

3.0 Page Previewer
<table>
<thead>
<tr>
<th>File Name</th>
<th>File Size</th>
<th>Processed Data Size</th>
<th>Comp Method</th>
<th>Compressed File Size</th>
<th>Percent Reduct</th>
<th>Pgs</th>
<th>Page Data</th>
<th>Compressed Page Data</th>
<th>Percent Reduct</th>
<th>Patt</th>
<th>Patt Data</th>
<th>Compressed Patt Data</th>
<th>Percent Reduct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slide</td>
<td>22,160</td>
<td>21,162</td>
<td>RLE</td>
<td>20,288</td>
<td>4.1</td>
<td>2</td>
<td>3,733</td>
<td>3,540</td>
<td>5.2</td>
<td>237</td>
<td>17,425</td>
<td>16,744</td>
<td>3.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SRLE</td>
<td>20,001</td>
<td>5.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Huff</td>
<td>14,858</td>
<td>29.8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>LZW</td>
<td>13,222</td>
<td>37.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PKARC</td>
<td>12,909</td>
<td>40.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PKZIP</td>
<td>10,209</td>
<td>51.8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Slide5</td>
<td>35,280</td>
<td>33,509</td>
<td>RLE</td>
<td>32,637</td>
<td>2.6</td>
<td>6</td>
<td>16,098</td>
<td>15,907</td>
<td>1.2</td>
<td>238</td>
<td>17,407</td>
<td>16,726</td>
<td>3.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SRLE</td>
<td>32,374</td>
<td>3.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Huff</td>
<td>24,816</td>
<td>25.9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>LZW</td>
<td>18,552</td>
<td>44.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PKARC</td>
<td>17,605</td>
<td>47.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PKZIP</td>
<td>10,615</td>
<td>68.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Slide10</td>
<td>52,080</td>
<td>49,253</td>
<td>RLE</td>
<td>48,391</td>
<td>1.8</td>
<td>11</td>
<td>31,544</td>
<td>31,363</td>
<td>0.6</td>
<td>243</td>
<td>17,695</td>
<td>17,014</td>
<td>3.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SRLE</td>
<td>48,148</td>
<td>2.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Huff</td>
<td>36,594</td>
<td>25.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>LZW</td>
<td>23,370</td>
<td>52.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PKARC</td>
<td>22,566</td>
<td>54.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PKZIP</td>
<td>11,023</td>
<td>77.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Schedule</td>
<td>25,520</td>
<td>24,247</td>
<td>RLE</td>
<td>23,932</td>
<td>1.3</td>
<td>2</td>
<td>15,258</td>
<td>15,076</td>
<td>1.2</td>
<td>204</td>
<td>8,985</td>
<td>8,852</td>
<td>1.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SRLE</td>
<td>23,887</td>
<td>1.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Huff</td>
<td>18,649</td>
<td>23.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>LZW</td>
<td>14,359</td>
<td>40.8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PKARC</td>
<td>13,430</td>
<td>44.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PKZIP</td>
<td>10,720</td>
<td>55.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 3-1: Compression Results Continued

<table>
<thead>
<tr>
<th>File Name</th>
<th>File Size</th>
<th>Processed Data Size</th>
<th>Comp Method</th>
<th>Compressed File Size</th>
<th>Percent Reduct</th>
<th>Compressed Page Data</th>
<th>Percent Reduct</th>
<th>Patt</th>
<th>Patt Data</th>
<th>Compressed Patt Data</th>
<th>Percent Reduct</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Resume</strong></td>
<td>61,360</td>
<td>57,712</td>
<td>RLE</td>
<td>57,442</td>
<td>0.5</td>
<td>48,753</td>
<td>0.4</td>
<td>130</td>
<td>8,975</td>
<td>8,896</td>
<td>0.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SRLE</td>
<td>57,437</td>
<td>0.5</td>
<td>48,598</td>
<td>0.3</td>
<td>8,833</td>
<td>1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Huff</td>
<td>44,106</td>
<td>23.6</td>
<td>36,348</td>
<td>25.4</td>
<td>5,852</td>
<td>34.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>LZW</td>
<td>30,841</td>
<td>46.6</td>
<td>23,674</td>
<td>51.4</td>
<td>5,552</td>
<td>38.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PKARC</td>
<td>32,264</td>
<td>44.1</td>
<td>26,357</td>
<td>45.9</td>
<td>5,726</td>
<td>38.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PKZIP</td>
<td>23,710</td>
<td>58.9</td>
<td>19,108</td>
<td>60.8</td>
<td>4,629</td>
<td>48.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Turbo2</strong></td>
<td>20,960</td>
<td>19,823</td>
<td>RLE</td>
<td>19,279</td>
<td>2.7</td>
<td>11,134</td>
<td>3.9</td>
<td>125</td>
<td>8,332</td>
<td>8,141</td>
<td>1.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SRLE</td>
<td>13,221</td>
<td>3.0</td>
<td>11,127</td>
<td>4.0</td>
<td>8,090</td>
<td>1.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Huff</td>
<td>15,516</td>
<td>21.4</td>
<td>9,016</td>
<td>22.2</td>
<td>5,566</td>
<td>32.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>LZW</td>
<td>11,818</td>
<td>40.4</td>
<td>5,901</td>
<td>49.1</td>
<td>5,149</td>
<td>37.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PKARC</td>
<td>11,061</td>
<td>44.2</td>
<td>5,840</td>
<td>49.6</td>
<td>5,123</td>
<td>37.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PKZIP</td>
<td>8,580</td>
<td>55.7</td>
<td>4,374</td>
<td>62.3</td>
<td>4,232</td>
<td>48.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Turing</strong></td>
<td>78,320</td>
<td>73,934</td>
<td>RLE</td>
<td>73,290</td>
<td>0.9</td>
<td>46,961</td>
<td>0.4</td>
<td>340</td>
<td>26,969</td>
<td>26,515</td>
<td>1.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SRLE</td>
<td>72,993</td>
<td>1.3</td>
<td>46,815</td>
<td>0.3</td>
<td>26,174</td>
<td>3.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Huff</td>
<td>74,167</td>
<td>26.7</td>
<td>32,762</td>
<td>30.2</td>
<td>16,947</td>
<td>37.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>LZW</td>
<td>43,646</td>
<td>41.0</td>
<td>22,778</td>
<td>51.5</td>
<td>16,671</td>
<td>38.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PKARC</td>
<td>41,384</td>
<td>44.0</td>
<td>24,678</td>
<td>47.5</td>
<td>16,857</td>
<td>37.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PKZIP</td>
<td>35,389</td>
<td>52.1</td>
<td>21,222</td>
<td>54.8</td>
<td>14,189</td>
<td>41.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>49,280</td>
<td>46,807</td>
<td>RLE</td>
<td>39,371</td>
<td>1.6</td>
<td>24,845</td>
<td>0.9</td>
<td>217</td>
<td>15,098</td>
<td>14,698</td>
<td>2.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SRLE</td>
<td>39,152</td>
<td>2.0</td>
<td>24,644</td>
<td>0.8</td>
<td>14,504</td>
<td>3.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Huff</td>
<td>29,824</td>
<td>25.3</td>
<td>18,205</td>
<td>26.7</td>
<td>9,749</td>
<td>35.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>LZW</td>
<td>27,258</td>
<td>44.3</td>
<td>11,467</td>
<td>53.8</td>
<td>9,245</td>
<td>38.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PKARC</td>
<td>21,604</td>
<td>45.9</td>
<td>12,190</td>
<td>50.9</td>
<td>9,288</td>
<td>38.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PKZIP</td>
<td>15,749</td>
<td>60.6</td>
<td>8,145</td>
<td>67.2</td>
<td>7,631</td>
<td>49.5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3.4.1 Run Length Encoding

Run length encoding (RLE) [20] replaces a running sequence of identical bytes with a three byte sequence consisting of a run length flag character, the repeated byte, and a repeat count. RLE executes quickly and the algorithms tend to be simple and short. RLE can generate compression ratios less than one if the flag character is part of the data being compressed. Since the PMP data covers the complete binary range, the flag character must be chosen carefully to maximize the compression ratio. A character frequency analysis was performed to determine a suitable flag character. Hexadecimal ‘AA’ was chosen to further determine the effectiveness of using RLE on the PMP data. The alternating binary ones and zeros within the bit pattern of ‘AAh’ are not likely to occur within the relatively large font bit maps being used by the previewer. Furthermore, ‘AAh’ is not a PMP command code. The results in Table 3-1 indicate that RLE is not effective for PMP data.

A run length frequency analysis was also performed which verified the results in Table 3-1. The run length frequency results for some PMP files are shown in Table 3-2. Table 3-2 indicates that shorter runs are more probable, which is expected. A shorter run length format could take advantage of the large number of three byte sequences. A shorter format can be implemented if the number of flag characters is increased to cover a consecutive range of input numbers. The repeat count may then be included within the flag character. The number of flag characters in use should be a power of two and would represent the longest sequence that could be encoded in the two byte format. Table 3-1 shows the results of using a short RLE format. The two byte format used a flag character range from ‘B0h’ through ‘BFh’ which was determined to be a fairly inactive range, for PMP data, from the character frequency
analysis. The two byte format supports run lengths of up to fifteen characters. Longer run lengths revert back to the normal three byte RLE format. The results are generally better than normal RLE but still indicate that PMP data is not effectively compressed by RLE.

<table>
<thead>
<tr>
<th>Run Length</th>
<th>Slide</th>
<th>Slide5</th>
<th>PMP Filename</th>
<th>Slide10</th>
<th>Turing</th>
<th>Turbtest</th>
<th>Resume</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>152</td>
<td>155</td>
<td>225</td>
<td>296</td>
<td>45</td>
<td>99</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>25</td>
<td>25</td>
<td>25</td>
<td>56</td>
<td>12</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>26</td>
<td>25</td>
<td>25</td>
<td>19</td>
<td>5</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>10</td>
<td>5</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>58</td>
<td>58</td>
<td>58</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>46</td>
<td>46</td>
<td>46</td>
<td>8</td>
<td>8</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>16+</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>14</td>
<td>9</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

Total 343 375 415 422 91 138

Avg 5.5 5.3 5.1 4.5 9.1 5.0
3.4.2 Huffman Compression

The idea behind Huffman compression [20] is to assign the most frequently occurring characters to shorter bit length codes. As a character’s frequency of occurrence diminishes, the length of its code increases. Instead of representing each character by an eight bit code, the most frequently occurring characters are represented by less than eight bits, and less frequently occurring characters are coded by more than eight bits. The average code length will be less than eight bits and so file compression will occur. The character codes are developed by constructing a Huffman tree from each character’s frequency of occurrence. Algorithms to accomplish this task may be found in references [20] and [23]. Once a tree has been constructed, characters are easily coded and decoded by tracing the tree. Huffman codes contain a prefix property which simplifies decompression. The prefix property ensures that no code is repeated from the beginning of a longer code.

A first order approximation of the effectiveness of using Huffman compression techniques may be obtained by calculating the entropy of a data stream from the frequency of occurrence of each character. Entropy, in information theory, is the average number of bits required to represent a character (bits/character) and is given by the equation:

\[ S = - \sum_{i=0}^{n} P_i \log_2 P_i \]  

(Eq. 3-1)

\( P_i \) is the probability or frequency of occurrence of the \( i \)'th symbol in an ‘n’ symbol alphabet. For eight bit characters, the alphabet would consist of 256 symbols and each symbol defined within a data stream would have a certain probability of occurrence. Subtracting the entropy from eight would then give an indication of the amount of redundancy
within the data stream in bits/byte. The percent reduction ($R_p$) may then be calculated by:

$$R_p = \left(1 - \frac{S}{S} \right) \times 100$$

(Eq. 3-2)

The method outlined above for estimating the effectiveness of using statistical compression techniques is an approximation. The probability of character occurrence, used to calculate the entropy, is independent of the character ordering within the data stream. In other words, the probability of the next character within the data stream is independent of the previous character. Therefore, the entropy calculated for a data stream will be, in general, greater because of the unaccounted redundancy associated with multiple character combinations. The Huffman tree does not consider character combinations, so the entropy calculation should be an accurate estimate of compression using Huffman encoding. A program by Kas Thomas [24] was employed to find the estimates recorded in Table 3-1 for Huffman compression.

A major disadvantage of using Huffman compression, within the communication module, is the necessity for a character frequency analysis on each data set. In addition, the frequency analysis must be transmitted to the receiver before decompression can be accomplished. These problems can largely be avoided by using a fixed frequency table which the sender and receiver have agreed upon prior to transmission. For more accuracy, an adaptive Huffman encoding scheme may be implemented. An optimum compression ratio will not be obtained but might be approached through a careful statistical analysis of PMP data. Such an analysis was not undertaken due to the effectiveness of another statistical encoding technique, LZW.
3.4.3 Lempel-Ziv-Welch (LZW) Compression

LZW compression [21] is another statistical encoding method. But unlike Huffman compression, LZW takes into account, at least partially, the character ordering within a data stream. LZW compression operates by assigning codes to character strings which are stored in a string table. The first 256 codes are reserved for the basic eight bit character set, while higher number codes are assigned to character strings. As a new character within the data stream arrives for compression it is added to the end of the current string. The current string consists of one or more characters that has an entry in the string table. A one character string always has an entry in the string table. The string table is searched and if the new string is also in the string table, then the new string becomes the current string and another data stream character is fetched. If the new string is not in the string table, then the new string is assigned the next available code and placed in the string table. Additionally, the code for the current string is output and the new character becomes the current string. The following pseudo-code describes the LZW compression algorithm:

```c
unsigned char string[256], ch;
string = getchar();
while((ch = getchar()) != EOF)
    if((string + ch) is in string table)
        string = string + ch;
    else {
        place (string + ch) in string table
        output code for string;
        string = ch;
    }
output code for string;
```

The implication of having codes greater than 256 is that each code output is at least nine bits long. So data streams that are short or have a very small number of redundant strings may actually be expanded instead of compressed. An LZW compression algorithm which uses variable
length codes will output nine bit codes until all 512 codes have been assigned, at which point the algorithm switches to ten bit codes. The switch is communicated to the decompressor by specifically assigning a code for that function. Since strings are assigned codes and placed in the string table before they are used for compression, the decompression algorithm can expand the character stream by rebuilding the original string table as the codes arrive.

The results for Table 3-1 were obtained by implementing an LZW routine into the page previewer. An LZW routine, originally written by Mark R. Nelson [21] for file compression and revised by Shawn M. Regan [22], was rewritten to handle asynchronous input for the communication module. Table 3-1 shows that the LZW routine's performance is similar to PKARC. The table also shows that if the string table is cleared after sending the pattern data, then the compression ratio can be improved. The page previewer clears the string table prior to sending the page data when the whole document is being transferred to the PC. LZW compression algorithms must search a large string table which is expensive in terms of memory usage and computation time. The decompression routine is not as hindered in computation time since it is not required to search a string table. As a result, LZW compression would be impractical for the PC which requires a responsive user interface. The computation time required to search the string table, for transferring data to the mainframe, may be too costly.
CHAPTER 4
FURTHER DEVELOPMENT

A major goal of the communication platform is to provide support for
distributed applications in order to minimize communication overhead by
utilizing the strengths of both machines. At present, the communication
platform operates with a commercial communication package as the termi-
nal/emulator command (TEC) module. The communication module’s routines
are presently an integral part of the page previewer application. The
direction and philosophy of the communication software should be the
separation of the communication platform from the application during
development. The application programmer should have a "clean" interface
to the communication platform so that development may concentrate on the
function of each program within an application. Future developments and
improvements to the communication platform should keep this basic tenet
to ensure that code will remain manageable. Improvements and extensions
to the communication platform will be discussed separately from the page
previewer. A section on other applications will also be included.
4.1 Communication Platform Extensions

The next major step in the evolution of the communication platform is to replace the HyperACCESS/5 communication package with a new terminal/emulator command (TEC) module. A replacement is necessary because of the monetary cost to the end user and the limited flexibility in controlling the amount of memory available to PC applications. Most communication packages are designed to handle terminal emulation as well as file transfer, which adds overhead beyond what the communication platform requires. File transfer could be considered as just another application, much as the page previewer is an application. But terminal emulation is a necessary aspect of the communication platform, even if the output is redirected to an application. The development of a new TEC module should also include terminal emulation and the ability to interpret mainframe output which is currently accomplished with a scripting language. In addition, the TEC must be able to launch application programs. Other additions to the TEC, such as file transfer, should be handled as separate applications. To minimize the memory consumption of the TEC, support for different terminals could be handled through interpreter tables located on disk or through the use of overlays.

A distributed application containing multiple programs, such as the page previewer, cannot take advantage of prior data sets to reduce the retransmission of redundant data. For example, the page previewer could be used many times during an editing session but a vast majority of page data may never change and should not require retransmission. The development of a database module, as discussed in Chapter 2, for the communication platform could be a useful addition for the reduction of communication. In addition the database module could eventually include routines for memory management and the use of virtual memory. Such additions could provide applications with a standard method of avoiding

4.0 Further Development
out of memory conditions while further insulating the programmer from machine specific details.

The communication module was the central focus of the current work and was specifically implemented and tailored for the page previewer application within a particular interconnection network. Extensions to the communication module should include the development of libraries for the support of other applications and networks. Details of the network and the communication module's configuration could be specified prior to compilation and the correct supporting routines included during program linkage. The next step, apart from using libraries, is to explore the idea of having the communication module become an integral part of the machine's operating system through the use of device drivers on the PC and as a nucleus extension on the mainframe. Further enhancements to the communication module include a library of compression algorithms and improved error handling. The communication module should be expanded above the DLC layer to include functions normally associated with the transport layer. In particular, functions are needed to handle errors which the DLC is unable to reconcile. These faults include duplicate or lost packets and flow control problems.

4.0 Further Development
4.2 Page Previewer Extensions

Extensions to the page previewer include optimizing and augmenting the present code. Various commands are not supported by the PMP interpreter. Appendix 'B' lists the support level of each PMP command. It is uncertain whether the complete set of PMP commands, apart from the printer interrogation commands, are required to preview SCRIPT documents. For example, some implemented rotation commands have not been supported by the SCRIPT/GML system at our installation. The limitations and output stream of SCRIPT should be examined when future extensions to the PMP interpreter are considered. In general, the limitations and output stream of each object producing program should be examined to prioritize which PMP commands should be supported.

The display performance of the PMP interpreter may be improved. Initial profiling indicated that the most serious bottleneck was the time required for screen output. The page previewer uses an intermediate graphics library for screen output which adds an extra layer of software to the output routines. Writing directly to video memory, along with new pattern and vector output algorithms, may increase performance. The PMP interpreter would also benefit by rewriting key algorithms in assembly language.

The page previewer’s code also requires revision to support large PMP document files. Memory management could be accomplished by storing the most recently or frequently used pages and fonts in memory. Extra fonts and pages could either be requested from the mainframe, on an as needed basis, or stored on disk for quicker retrieval. Memory management for the page previewer could make use of the database module described at the end of Chapter 2.

4.0 Further Development
Implementation of a set of local font files would also decrease the amount of serial communication required, while helping the page previewer display pages quicker. For example, the file "slide" requires seven seconds to transfer two pages of data at 19,200 baud. If the pattern data was locally available the transmission time would drop to less than 1.3 seconds; assuming a standard pattern numbering scheme that requires minimal communication overhead. To further improve the performance of such a scheme the font patterns should be stored in a compressed format to reduce the slow process of reading the disk. Additionally, the font files should include an indexing format to decrease the seek time necessary to find a particular font and pattern on the disk. A disk cache could also prove helpful, but a local pattern manager could prove more useful by controlling the pattern availability within the same block of memory used to store the patterns accessed by the PMP interpreter. Implementing local font files would simplify the remote page/pattern manager which is unable to associate patterns with a particular page. Local font files may alleviate the need to associate patterns with pages.

Document presentation on the PC screen could be improved. The scaling routines use a simple row and column deletion algorithm which subtracts more information than is necessary from the original bit map. As a result, the readability of some documents is diminished. The ability to scale a full line of legible text across the screen would make the previewer much easier to use. Reference [14] provides a source for algorithms which may be adapted to the previewer's scaling function to improve pattern resolution. Ideas and insights to improve the page previewer's user interface may also be obtained from references [13, 15, 16, 18].

4.0 Further Development
The page previewer could also be used as a previewer for SCRIPT documents that will be printed on postscript printers. The page previewer will continue to process and preview PMP data but the output will not be true WYSIWYG previewing, although early results indicate that the differences are very minute. A PMP interpreter has the advantage of being much smaller and faster than current postscript previewers and therefore would be more suitable in an integrated word processing environment.
4.3 Other Applications

Other distributed applications can also use the communication platform. An editor application which is directly related to, and could be included with, the page previewer would vastly improve the mainframe editor, VM/XEDIT. Such a full screen editor could include pull down menus, a mouse, easier scrolling, graphics capabilities, and improved integration of the page previewer program. For operation over low speed serial lines the editor's user interface could be located on the PC while background communication sends editing information and page data to the mainframe. The remote part of the editor could then build a mirror image of the document in the mainframe's memory for the purpose of quickly running spelling and grammar checkers along with the page previewer for document processing. Such an editor could be used in an iterative programming application with multiple compilers and a debugger. In either case, the power of the mainframe could be focused on the computationally intensive sections of the application while the PC provides the user with a comfortable graphical user interface (GUI).

An application of particular interest is one that would "merge" the operating systems of both the PC and mainframe. The user could be saved from learning two operating systems while the operating system application translates user commands into an appropriate native syntax. An operating system application could replace the TEC module and provide a GUI environment for mainframe sessions. Such an application could transform the PC into a more efficient work station. The application could perform like the X/Windows system but would operate differently. In X/Windows, a powerful host controls most of the window information and requires a high bandwidth communication path. But unlike X/Windows and similar to the page previewer, the operating system application would locate its GUI locally to the user in order to minimize communica-
tion and would operate the mainframe as a server. A disadvantage to such a role reversal for reducing communication overhead, relative to X/Windows, is that more memory and processing time must be allocated to the user interface on the local machine.
CHAPTER 5
CONCLUSION AND RECOMMENDATIONS

Elements of a communication platform have been presented which allow transparent communication between distributed applications over existing 3270 connections. The communication platform is oriented toward supporting and optimizing PC to IBM mainframe interaction over low throughput serial lines (19,200 baud or less), which are controlled by the mainframe through a protocol converter. The communication platform is a direct extension of the file transfer methodology which has been used in the past to improve interaction between the PC and IBM mainframe. A simple data link control (DLC) protocol has been applied within each side of a distributed application as a first step toward a general communication platform for the 3270 environment.

The communication services have been presented through the continued development of a distributed WYSIWYG page previewer for SCRIPT. In the development and optimization of this particular application, the conclusion may be drawn that improved performance is largely application dependent. The communication platform provides an interface for both sides of an application to communicate, however, it is the responsibility of the application to use that interface to its full potential. How the application is written directly affects its performance. While the communication platform can provide services bent towards supporting decreased communication, it is up to the application programmer to implement those services relative to the problem being addressed. In the case of the page previewer, communication overhead was decreased through the preprocessing of PMP data, physical compression optimized for the PMP data, and background communication; all of which are application dependent.

5.0 Conclusion
The original SCRIPT previewer code was PC based and operated using file transfers under Kermit. To ease the transition to a distributed application while using the communication platform, the user interface was improved to support background communication. Larger viewing windows were also added which accommodate scaling, panning, and rotating of document images using both the keyboard and mouse. In addition, the PMP interpreter was rewritten with new data structures and algorithms for faster displays and support of all the PMP orientation commands. Improved error support was added and a relative decrease in code size was obtained. Table 5-1 summarizes the performance results of replacing Kermit with the communication module. The communication module transfers the necessary PMP data, on average, 6.9 times faster than Kermit.

<table>
<thead>
<tr>
<th>Table 5-1: PMP File Transfer Time At 2400 Baud</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Name</td>
</tr>
<tr>
<td>-----------</td>
</tr>
<tr>
<td>Resume</td>
</tr>
<tr>
<td>Turing</td>
</tr>
<tr>
<td>Average</td>
</tr>
</tbody>
</table>

Notes:
1) Transfer Factor = (Kermit transfer time)/(previewer transfer time)
2) [time] = Page previewer transfer time at 19,200 baud.
3) Transfer times include the cover page which is approximately 620 bytes without its pattern data.

5.0 Conclusion
It must be acknowledged that the communication platform was developed within a restricted environment (LU 2) which is slowly being replaced by peer-to-peer communication through IBM's commitment to the LU 6.2 protocol (APPC). The LU 6.2 protocol defines option subsets which support various communication functions between two programs [9]. It is not clear whether a basic set of the LU 6.2 protocol could be efficiently implemented on a PC with limited memory. Therefore a smaller, layered protocol could be developed, from the ideas presented in this paper, to support PC to mainframe program-to-program communication. It is recommended that the LU 6.2 protocol be examined for inclusion into a PC operating system to support distributed PC/mainframe applications.

5.0 Conclusion
BIBLIOGRAPHY

Communication


Page Preview


Compression


Programming


APPENDIX A

7171 ASCII ATTACHMENT CONTROL UNIT

The 7171 ASCII attachment control unit [37] allows an asynchronous ASCII terminal to appear as a 3278 terminal to the mainframe. The 7171 appears as one or two 3274 control units to the host and supports up to 64 users. The conversion from and to a 3270 environment to an ASCII environment involves the translation of data streams, character sets, and the network's physical layer. The 7171 must convert the 3270 data stream [36] into the escape sequences of the connected ASCII terminal. In addition, the EBCDIC character set must be mapped to the ASCII character set. Character translation can be a problem for transferring files over such a heterogeneous network. The 3270 environment uses the BISYNC (binary synchronous communications) or SDLC (synchronous data link control) protocol whereas ASCII terminals are usually connected asynchronously. The BISYNC protocol is a half duplex character oriented protocol whereas SDLC is a full duplex bit oriented protocol. Viewed from the ASCII terminal, the 7171 operates in a half duplex mode even though the asynchronous line may be full duplex. This may be due to BISYNC being an early and major protocol within the 3270 environment.
Transparent Mode

The 7171 supports two transparent modes which disable the EBCDIC to ASCII translation and the 3270 data stream conversion. The 3270 data stream is terminal oriented and so communication is limited to screen size data buffers. An invalid 3270 addressing command places the 7171 in a transparent mode and causes the remainder of the data stream, up to the buffer limit, to be transmitted without any data translation. Transparent data is treated as seven bit ASCII. Each transparent buffer must begin with the invalid addressing command to prevent protocol conversion.

A transparent write mode is supported which ignores all input from the ASCII terminal except the attention keys ENTER, PFK, and PA. Pressing these keys will cause the AID (Attention IDentifier) value to change. An AID is returned to the host after the 7171 processes each output data stream. The AID value will not change if the host did not unlock the keyboard upon issuing the transparent write. A master reset sequence (defined in the active terminal definition table) from the keyboard or a non-transparent write will end transparent mode. The transparent write mode is suitable for output equipment, such as printers and graphic devices, that cannot relate to the 3270 data streams.

The transparent write/read mode is similar to the transparent write mode but data from the ASCII terminal may be sent to the host after every transparent write. All input from the ASCII terminal is ignored while the 7171 transmits the output buffer. The 7171 interrupts the host when the input data stream is terminated by either a carriage return, a master reset sequence, or when the input buffer fills. The pacing stop character (defined in the active terminal definition table) may also terminate input depending on the 7171’s configuration. Each data char-

A: 7171 Protocol Converter 77
acter returned to the host has its most significant bit set. Transparent write/read is intended to support file transfer operations and "limited interactive functions."

Transparent Mode Limitations

The above description reveals some of the restrictions involved in using transparent mode. Besides the logical half duplex problem, the 7171 imposes limitations on character size, the interpretation of characters, and buffer lengths. While using transparent mode, only seven bit data may be transmitted and received. Manipulation of the parity bit to transfer binary data through the 7171 is prevented. The 7171 may be configured to transmit eight bit data. But the description of transparent mode given by IBM simply states that all input will have the most significant bit set and all output is treated as seven bit ASCII. A recommendation is also given to set the most significant bit of every output character. Obviously, IBM did not intend the transparent protocol to send eight data bits. Therefore, whether a 7171 configured to transmit eight bits could actually do so in transparent mode is uncertain.

Transparent communication to the mainframe is normally terminated by a carriage return character. The implication is that no carriage return character should occur within a packet. Similarly, the master reset sequence, and possibly the pacing character, should not be placed within data buffers. When these characters must be transmitted as data, a character oriented protocol will translate the characters to a "safe" value that will not be removed or interpreted by the intervening communication equipment. A prefix character will then be inserted in the data stream to mark the translation. A bit oriented protocol circumvents this problem by inserting individual bits into the data stream to avoid sequences that will cause difficulties. The 7171 imposes character limitations only on the input data stream.
Transparent communication from the mainframe will transmit all seven bit ASCII characters intact. One exception occurs while transmitting the XOFF pacing character. If XOFF is the last character within an output buffer then it is not transmitted. If the XOFF character appears anywhere else within the buffer then it will be transmitted normally. To prevent lost XOFF characters, the remote communication module appends an extra character to the end of each packet. The extra character should not be the 7171’s defined XOFF character.

The amount of application data transmitted within a packet is limited by the 7171’s buffer size. The maximum buffer size depends upon the terminal model recognized by the 7171 and the command operation used to invoke transparent mode. The default buffer length is 1920 bytes but may be extended to 3565 bytes. The full buffer length is not available to the packet. Each buffer requires a seven byte sequence specifying transparent mode. A bug or anomaly exists in the 7171 firmware which may prevent the proper transmission of a full transparent buffer depending upon the last character. The remote communication module employs a buffer which is one less than the maximum to avoid this particular bug. Data input to the mainframe should allow a three byte overhead to prevent overrunning the 7171 buffer. These three bytes are used to store the returned AID byte along with a buffer cursor address.
APPENDIX B
PAGE PREVIEWER PMP COMMANDS

Page map primitive (PMP) commands are used to drive an IBM 3812 page printer. The page previewer's 3812 interpreter supports a subset of the PMP commands. This appendix details the support given by the page previewer for the commands listed in reference [29]. Additional PMP commands which are listed in more comprehensive references are not supported and will generate error messages from the interpreter. Table B-1 categorizes the supported PMP commands and lists an appropriate support level. The following emulation notes further explain the support available for some instructions and may be cross listed from Table B-1.

00 - 7F: Generate Font Patterns - No modified spacing is supported. An error message will be generated if the pattern cannot be found within the active font. Support for printing an alternate or error pattern is not supported.

C2: Deactivate Font - This command is used to prevent the active and alternate fonts from being internally rotated to match a new page orientation when they will not be subsequently used in that orientation. The page previewer does not internally rotate fonts to match the page orientation and so this command is ignored.
<table>
<thead>
<tr>
<th>Command Set</th>
<th>Opcode</th>
<th>PMP Command</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page parameter</td>
<td>D1</td>
<td>Print Page</td>
<td>Full</td>
</tr>
<tr>
<td></td>
<td>L2</td>
<td>Set Page Orientation</td>
<td>Full</td>
</tr>
<tr>
<td></td>
<td>F6</td>
<td>Set Page Size.</td>
<td>Partial</td>
</tr>
<tr>
<td>Cursor control</td>
<td>E2</td>
<td>Move cursor horizontally.</td>
<td>Full</td>
</tr>
<tr>
<td></td>
<td>E3</td>
<td>Move cursor vertically.</td>
<td>Full</td>
</tr>
<tr>
<td></td>
<td>9x</td>
<td>Restore cursor.</td>
<td>Full</td>
</tr>
<tr>
<td></td>
<td>E4</td>
<td>Restore cursor component.</td>
<td>Full</td>
</tr>
<tr>
<td></td>
<td>8x</td>
<td>Save cursor.</td>
<td>Full</td>
</tr>
<tr>
<td></td>
<td>E0</td>
<td>Set cursor horizontally.</td>
<td>Full</td>
</tr>
<tr>
<td></td>
<td>E1</td>
<td>Set cursor vertically.</td>
<td>Full</td>
</tr>
<tr>
<td>Font management</td>
<td>D7</td>
<td>Activate Alternative Font</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>D3</td>
<td>Activate Font</td>
<td>Full</td>
</tr>
<tr>
<td></td>
<td>E5</td>
<td>Copy Font</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>FA</td>
<td>Copy Font Pattern</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>C2</td>
<td>Deactivate Font</td>
<td>Ignored</td>
</tr>
<tr>
<td></td>
<td>F0-F3</td>
<td>Load Font Pattern</td>
<td>Full</td>
</tr>
<tr>
<td></td>
<td>F4</td>
<td>Load Large Font Pattern</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>C7</td>
<td>Unload All Fonts</td>
<td>Ignored</td>
</tr>
<tr>
<td></td>
<td>D4</td>
<td>Unload Font</td>
<td>Ignored</td>
</tr>
<tr>
<td>Pel generation</td>
<td>00-7F</td>
<td>Generate Font Patterns</td>
<td>Partial</td>
</tr>
<tr>
<td></td>
<td>F8</td>
<td>Generate Pattern Immediate</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>F9</td>
<td>Generate Vectors</td>
<td>Partial</td>
</tr>
<tr>
<td></td>
<td>D8</td>
<td>Generate Vectors - Close &amp; Fill</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>D9</td>
<td>Set Font Pattern Controls</td>
<td>Partial</td>
</tr>
<tr>
<td></td>
<td>D0</td>
<td>Set Generation Mode</td>
<td>Partial</td>
</tr>
<tr>
<td>Macro</td>
<td>FB</td>
<td>Execute Library Macro</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>DA</td>
<td>Execute Macro</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>A0-BF</td>
<td>Execute Macro - Short</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>F7</td>
<td>Load Macro</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>C8</td>
<td>Unload All Macros</td>
<td>Ignored</td>
</tr>
<tr>
<td>Device Control</td>
<td>C6</td>
<td>Jog Exit Tray</td>
<td>Ignored</td>
</tr>
<tr>
<td></td>
<td>C5</td>
<td>Ring Bell</td>
<td>Ignored</td>
</tr>
<tr>
<td></td>
<td>D6</td>
<td>Set Display</td>
<td>Ignored</td>
</tr>
<tr>
<td></td>
<td>C0</td>
<td>Stop</td>
<td>Ignored</td>
</tr>
<tr>
<td>Query</td>
<td>C1</td>
<td>Query Device</td>
<td>Ignored</td>
</tr>
<tr>
<td></td>
<td>FC</td>
<td>Query File</td>
<td>Ignored</td>
</tr>
<tr>
<td></td>
<td>C3</td>
<td>Query Library</td>
<td>Ignored</td>
</tr>
<tr>
<td></td>
<td>D5</td>
<td>Query Pattern</td>
<td>Ignored</td>
</tr>
</tbody>
</table>

Full = Complete emulation support is available for this command.
Ignored = Emulation within the page previewer is not needed.
None = The PMP command is not supported.
Partial = Not all options are available for this command.
D2: **Set Page Orientation** - All the page orientations are supported but the emulation of this command does not match the IBM 3812 page printer. The page printer rotates the active and alternate fonts when this command is issued. The interpreter does not rotate fonts to save processing time but uses an effective orientation which affects the printing algorithm. Refer to Appendix 'C' for more information on effective orientations.

D7: **Activate Alternative Font** - This command specifies an alternate font whose patterns are printed in case the active font is missing a pattern. The page printer always assumes that the PMP data stream will reference a pattern that is present in the active font. The page previewer will generate an error message for this command. If a referenced font pattern is not in memory then the page previewer will generate an error message.

D8: **Set Font Pattern Controls** - The only printing direction supported is "left to right". The cursor line is determined solely by the cursor offset parameter. No modified spacing is supported.

D9: **Set Generation Mode** - Only the default or power-on state is supported. The page map is always cleared to an all white background prior to page printing. A '1' bit is always printed as a dark pixel while a '0' bit is ignored.

E4: **Restore Cursor Component** - The PMP data stream uses two operands to specify the register component which should be restored. The previewer stores the data in one operand. The interpreter generates error messages for invalid component and register values.
F6: **Set Page Size** - Only 8 1/2" x 11" paper is supported. Smaller page sizes may be displayed but on a white 8 1/2" x 11" background. This command clears the page map and resets the cursor registers on a 3812. The page previewer always clears the page map and cursor registers prior to printing a page. The assumption is also made that this command will never occur after data has already been generated on the page map. Other than checking the paper height limit, this command is ignored.

F8: **Generate Vectors** - Diagonal vectors are not supported. Documents which contain diagonal vectors will be formatted properly but the diagonal vectors will not printed. Accurate emulation is not being accomplished for lack of technical data on page printer vector generation. For instance, the vector thickness is always adjusted to an odd value to ensure that connecting vectors are centered properly and round vector ends are not supported. Each vector end is also increased by one half the vector thickness.
APPENDIX C
PATTERN DISPLAY ALGORITHM

The font pattern display algorithm is used to interpret the "generate font patterns" command (1-66). The numbers within parentheses are page references for the first edition of the IBM 3812 Page Printer Programming Reference [29]. The pattern display algorithm does not support the modified spacing mode and assumes the printable pattern is in the active font.

The page map primitive (PMP) command set supports four page orientations (1-5): portrait, landscape right, portrait upside down, and landscape left. Each orientation defines the "top" and "left" side of a page. Table C-1 shows the orientation abbreviations and number assignments that are used throughout the PMP interpreter. Figure C-1 shows the coordinate systems' origins and orientations. For each coordinate system the "+x" axis represents the "top" of a page and the "+y" axis represents the "left" side of a page. The current printing position on a page is defined by an x-y coordinate called the cursor. The coordinates of the cursor change to reflect the current page orientation.

<table>
<thead>
<tr>
<th>Orientation</th>
<th>Abbreviation</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Portrait</td>
<td>P</td>
<td>0</td>
</tr>
<tr>
<td>Landscape Right</td>
<td>LR</td>
<td>1</td>
</tr>
<tr>
<td>Portrait Upside Down</td>
<td>PU</td>
<td>2</td>
</tr>
<tr>
<td>Landscape Left</td>
<td>LL</td>
<td>3</td>
</tr>
</tbody>
</table>
In addition to the PMP command stream specifying a page orientation, the page previewer supports viewing a page within the same four orientations independent of the PMP page orientation. The independence between the user selected screen orientation and the PMP orientation occurs because the PMP command stream can change the orientation many times within one page. The font patterns may also be loaded in any of the four orientations. For these reasons, automatically determining whether a page should be viewed as a portrait or as a landscape might prove difficult. An addition to the page previewer’s user interface might include an option for allowing the PMP interpreter to select the screen orientation.
A base coordinate system is defined, as shown in Figure C-1, to keep track of the PMP orientation and the screen orientation for printing a graphics screen. The user defined screen orientation is always placed over the base coordinate system whose orientation always corresponds to the upper left corner (Figure C-1). A display window coordinate system is defined with the same orientation as the base coordinate system but whose origin reflects that portion of a page currently being viewed on the screen. The display window's size, with respect to the full PMP page size in pels, depends upon the number of pixels assigned to the physical window multiplied by the viewing scale.

The cursor position employed while interpreting the PMP command set is always defined within the current PMP coordinate system. As a result, the PMP commands, such as 'move', 'set', 'save', and 'restore', do not need to check or reference any other coordinate system since 'x' is always considered horizontal and 'y' vertical. When the PMP page orientation is changed the cursor and cursor registers continue to point to the same physical page pel but their coordinates are changed to reflect the new page coordinate system (1-38). The coordinates are transformed by the equations shown in Table C-2. The equations in Table C-2 show a distinct pattern and can be reduced from sixteen equations per coordinate to four by assigning each orientation a number (Table C-1) and then calculating an effective orientation. The effective orientation may be calculated by the following the equation:

\[ O_e = (O_{new} - O_{old}) \& 3 \]  

(Eq. C-1)

The "&" symbol represents the bitwise AND operator. Table C-3 shows the reduced equation set necessary for coordinate transformations when using the effective orientation. These equations are employed to redefine the cursor position and cursor registers when a PMP page orientation command is issued.

C: Pattern Display
### Table C-2: Coordinate System Transformation Equations

<table>
<thead>
<tr>
<th>Old</th>
<th>New</th>
<th>New 'x' coord</th>
<th>New 'y' coord</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>P</td>
<td>$x_p = x_p$</td>
<td>$y_p = y_p$</td>
</tr>
<tr>
<td></td>
<td>LR</td>
<td>$x_{lr} = y_p$</td>
<td>$y_{lr} = x_{max} - x_p$</td>
</tr>
<tr>
<td></td>
<td>PU</td>
<td>$x_{pu} = x_{max} - x_p$</td>
<td>$y_{pu} = y_{max} - y_p$</td>
</tr>
<tr>
<td></td>
<td>LL</td>
<td>$x_{ll} = y_{max} - y_p$</td>
<td>$y_{ll} = x_p$</td>
</tr>
<tr>
<td>LR</td>
<td>P</td>
<td>$x_p = y_{max} - y_{ll}$</td>
<td>$y_p = x_{ll}$</td>
</tr>
<tr>
<td></td>
<td>LR</td>
<td>$x_{lr} = x_{lr}$</td>
<td>$y_{lr} = y_{lr}$</td>
</tr>
<tr>
<td></td>
<td>PU</td>
<td>$x_{pu} = y_{ll}$</td>
<td>$y_{pu} = x_{max} - x_{lr}$</td>
</tr>
<tr>
<td></td>
<td>LL</td>
<td>$x_{ll} = x_{max} - x_{lr}$</td>
<td>$y_{ll} = y_{lr}$</td>
</tr>
<tr>
<td>PU</td>
<td>P</td>
<td>$x_p = x_{max} - x_{pu}$</td>
<td>$y_p = y_{max} - y_{pu}$</td>
</tr>
<tr>
<td></td>
<td>LR</td>
<td>$x_{lr} = x_{max} - x_{pu}$</td>
<td>$y_{lr} = y_{pu}$</td>
</tr>
<tr>
<td></td>
<td>PU</td>
<td>$x_{pu} = x_{pu}$</td>
<td>$y_{pu} = y_{pu}$</td>
</tr>
<tr>
<td></td>
<td>LL</td>
<td>$x_{ll} = y_{pu}$</td>
<td>$y_{ll} = x_{max} - x_{pu}$</td>
</tr>
<tr>
<td>LL</td>
<td>P</td>
<td>$x_p = y_{ll}$</td>
<td>$y_p = x_{max} - x_{ll}$</td>
</tr>
<tr>
<td></td>
<td>LR</td>
<td>$x_{lr} = x_{max} - x_{ll}$</td>
<td>$y_{lr} = y_{ll}$</td>
</tr>
<tr>
<td></td>
<td>PU</td>
<td>$x_{pu} = y_{max} - y_{ll}$</td>
<td>$y_{pu} = x_{ll}$</td>
</tr>
<tr>
<td></td>
<td>LL</td>
<td>$x_{ll} = x_{ll}$</td>
<td>$y_{ll} = y_{ll}$</td>
</tr>
</tbody>
</table>

$x_{max} =$ Length of full page along x-axis of Old
$y_{max} =$ Length of full page along y-axis of Old

### Table C-3: Reduced Coordinate Transform Equations

<table>
<thead>
<tr>
<th>Os</th>
<th>New 'x' coordinate</th>
<th>New 'y' coordinate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>x</td>
<td>y</td>
</tr>
<tr>
<td>1</td>
<td>$x_{max} - x$</td>
<td>$y_{max} - y$</td>
</tr>
<tr>
<td>2</td>
<td>$y_{max} - y$</td>
<td>$x_{max} - x$</td>
</tr>
<tr>
<td>3</td>
<td>$x$</td>
<td>$y$</td>
</tr>
</tbody>
</table>

$x =$ Current 'x' coordinate in the old orientation.
y =$ Current 'y' coordinate in the old orientation.
x$_{max}$ = Length of full page along x-axis of Old
y$_{max}$ = Length of full page along y-axis of Old
The calculation of an effective orientation is also employed to handle the many combinations of PMP orientations, screen orientations, and pattern load orientations. Since the base coordinate system is always aligned with the screen orientation, an effective orientation can be calculated so that patterns and vectors may be displayed on the screen in their correct page orientation. An equation that accounts for all three orientation sources is given by:

\[ O_e = (\text{PMP} - (\text{Screen} + \text{Pattern})) \mod 3 \]  

(Eq. C-2)

This function is used to simplify and determine the correct set of equations and routines required for displaying patterns and vectors on the screen.

Printing a font pattern can be categorized into the operations of locating the pattern’s position on the page and then determining the pattern’s orientation. Both operations depend upon the current screen orientation, PMP orientation, and the pattern’s load orientation. In addition, locating the pattern’s position also depends upon the currently defined printing direction. The printing direction specifies how a series of successive patterns will be positioned relative to each other (1-7). The interpreter only supports the print direction "across", which specifies pattern spacing and placement for text read from the left to the right. To locate the pattern’s position on the page, the cursor positioning parameters (1-27) shown in Figure C-2 are used. Once the print point is located then the effective orientation is calculated to determine the correct routines for displaying the pattern. The overall pattern display algorithm may be outlined as follows:

1) Calculate the print-point using the current cursor location.
2) Transform the print-point into the base coordinate system.

C: Pattern Display
3) Use an area filter to describe the portion of a pattern that falls within the display window.

4) Size the pattern to the window scale by applying a size filter to the pattern.

5) Display the visible section of the pattern.

6) Calculate the end point which will become the new current cursor location.

The details of each step are presented next.

---

**Figure C-2:** Font Pattern Positioning Parameters

- **CLOFF** = Center line offset
- **LT** = Left/top space
- **PH** = Pattern height
- **PW** = Pattern width
- **RB** = Right/bottom space
1) Print-point Calculation

The PMP interpreter defines the print-point at that corner of the pattern box which corresponds to the start of the stored bit image. The patterns are stored in memory so that the pattern width is always an integral number of bytes. The actual width of the pattern, in bits, is padded with binary zeros. Each padded row is stored sequentially in memory from the print-point row (row 0) to the print height row. The PMP interpreter always references the print-point so that the display routines can read the font pattern sequentially from memory instead of calculating memory offsets or rotating the pattern to find a particular corner of the pattern.

The location of the print-point depends upon the current printing direction, as specified in the "set font pattern controls" command, and on the pattern load orientation. Equations to calculate the print-point coordinates for a particular print direction are shown in Table C-4. The equations in Table C-4 are only valid for a portrait pattern load orientation. Patterns may be loaded into memory in the four orientations shown in Figure C-3. The patterns in Figure C-3 are oriented with the print point located in the upper left hand corner. Note that the pattern width and height are defined with respect to the pattern storage as shown by the print-point. Because the pattern width and height are independent of the pattern character orientation, equations that depend on the pattern box dimensions should account for the pattern load orientation. Table C-5 shows the equations necessary to calculate the print-point location when the pattern load orientation is considered. The equations in Table C-5 are only valid for the "across" print direction. The equations in Table C-5 can be derived from Figure C-4. The orientation of the letter 'A' in Figure C-4 corresponds to the PMP orientation.
because the page printer will rotate the pattern to match the current PMP orientation.

### Table C-4

<table>
<thead>
<tr>
<th>Print Direction</th>
<th>$x_{pp}$</th>
<th>$y_{pp}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>across</td>
<td>$x + LT$</td>
<td>$y - CLOFF$</td>
</tr>
<tr>
<td>back</td>
<td>$x - (RB + PW)$</td>
<td>$y - CLOFF$</td>
</tr>
<tr>
<td>down</td>
<td>$x - CLOFF$</td>
<td>$y + LT$</td>
</tr>
<tr>
<td>up</td>
<td>$x - CLOFF$</td>
<td>$y - (RB + PH)$</td>
</tr>
</tbody>
</table>

### Table C-5

<table>
<thead>
<tr>
<th>Load Orientation</th>
<th>$x_{pp}$</th>
<th>$y_{pp}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>$x + LT$</td>
<td>$y - CLOFF$</td>
</tr>
<tr>
<td>LR</td>
<td>$x + LT$</td>
<td>$y - CLOFF + PW$</td>
</tr>
<tr>
<td>PU</td>
<td>$x + LT + PW$</td>
<td>$y - CLOFF + PH$</td>
</tr>
<tr>
<td>LL</td>
<td>$x + LT + PH$</td>
<td>$y - CLOFF$</td>
</tr>
</tbody>
</table>
Figure C-3: Pattern Load Orientations

The PMP interpreter is not strictly emulating an IBM 3812 page printer by handling pattern orientation through the location of a print-point. An advance function printer that uses the PMP command set will rotate the active font patterns to the current page orientation and then print the pattern. The overhead of rotating a font can be avoided by using pattern load orientations which correspond to the active page orientation (1-106). The purpose of the print-point is to avoid the need for pattern rotation and the calculation of row or column offsets.
Figure C-4: Print-point Position Relative to Load Orientation
2) Translating the Print-point into Base Coordinates

The calculation of the print-point presented above was relative to the current page orientation coordinate system. The print-point must be transformed to the base coordinate system so that the location of the pattern may be determined relative to the display window coordinate system. Since the print-point already reflects the load orientation, the following effective orientation equation may be used to transform the print-point into base coordinates:

$$O_x = (PMP - Screen) \& 3$$  \hspace{1cm} (Eq. C-3)

PMP and Screen refer to the current PMP and user interface screen orientations. The orientation values from Table C-1 are used in the equation. The translation equations are shown in Table C-6.

<table>
<thead>
<tr>
<th>$O_a$</th>
<th>$x_b$</th>
<th>$y_b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>$x_{pp}$</td>
<td>$y_{pp}$</td>
</tr>
<tr>
<td>LR</td>
<td>$x_{max} - x_{pp}$</td>
<td>$y_{max} - y_{pp}$</td>
</tr>
<tr>
<td>PU</td>
<td>$x_{max} - x_{pp}$</td>
<td>$y_{max} - y_{pp}$</td>
</tr>
<tr>
<td>LL</td>
<td>$y_{pp}$</td>
<td>$y_{pp}$</td>
</tr>
</tbody>
</table>

$x_{max}$ = Length of page along base x-axis.

$y_{max}$ = Length of page along base y-axis.
3) Area Filter

Given the print-point expressed in the base coordinate system, the area filter determines whether a font pattern is within the display window. The font pattern is positioned at the print-point in one of the four orientations shown in Figure C-5. The size and position of the window coordinate system, relative to the base coordinate system shown in Figure C-5, corresponds to that area of the document page that is being drawn on the screen. Although Figure C-5 shows the font pattern within the display window, the pattern may actually rest anywhere within the base coordinate system. Positioning information, pertaining to how much of the pattern is within the display window, is also determined and made available for the size filter through a common filter structure. The filter structure contains the number of pattern rows and columns, which are within the display window, and a row and column offset which represents the pattern's starting row and column numbers which are in the displayable window area. The following discussion summarizes the area filter algorithm.

The display window type is checked for the possibility that it encompasses the complete document page indicating that every pattern is within the display window. If the display window is full size, the filter structure is filled to indicate that the complete pattern is within the display window and control is then passed to the size filter algorithms.
A boundary check is made to determine whether the pattern falls within the display window. Since the pattern may be oriented in any of four orientations around the print-point (Figure C-5), Equation C-2 is used to calculate an effective orientation. The numbers below each pattern in Figure C-5 correspond to the effective orientation. Four boundary inequalities are associated with each effective orientation as shown in Table C-7. If any one of the four inequalities, for a given effective orientation, proves true then the font pattern is not within the display window. Such a condition causes the size filter and display algorithms to be skipped. The pattern load orientation is accounted for in Equation C-2, so that even though the inequalities in Table C-7 make use of
the pattern width and height no interchange of these quantities is necessary. The inequalities in Table C-7 are expressed more concisely in Tables C-8 and C-9.

### Table C-7

**Boundary Check Inequalities for the Display Window**

<table>
<thead>
<tr>
<th>$o_x$</th>
<th>Check</th>
<th>Inequality</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Top</td>
<td>$y_b + PH \leq y_{win}$</td>
</tr>
<tr>
<td></td>
<td>Right Side</td>
<td>$x_{win} + Scale \cdot Width_{win} \leq x_b$</td>
</tr>
<tr>
<td></td>
<td>Bottom</td>
<td>$y_{win} + Scale \cdot Height_{win} \leq y_b$</td>
</tr>
<tr>
<td></td>
<td>Left Side</td>
<td>$x_b + PH \leq x_{win}$</td>
</tr>
<tr>
<td>1</td>
<td>Top</td>
<td>$y_b + PW \leq y_{win}$</td>
</tr>
<tr>
<td></td>
<td>Right Side</td>
<td>$x_{win} + Scale \cdot Width_{win} \leq x_b - PH$</td>
</tr>
<tr>
<td></td>
<td>Bottom</td>
<td>$y_{win} + Scale \cdot Height_{win} \leq y_b$</td>
</tr>
<tr>
<td></td>
<td>Left Side</td>
<td>$x_b \leq x_{win}$</td>
</tr>
<tr>
<td>2</td>
<td>Top</td>
<td>$y_b \leq y_{win}$</td>
</tr>
<tr>
<td></td>
<td>Right Side</td>
<td>$x_{win} + Scale \cdot Width_{win} \leq x_b - PW$</td>
</tr>
<tr>
<td></td>
<td>Bottom</td>
<td>$y_{win} + Scale \cdot Height_{win} \leq y_b - PH$</td>
</tr>
<tr>
<td></td>
<td>Left Side</td>
<td>$x_b \leq x_{win}$</td>
</tr>
<tr>
<td>3</td>
<td>Top</td>
<td>$y_b \leq y_{win}$</td>
</tr>
<tr>
<td></td>
<td>Right Side</td>
<td>$x_{win} + Scale \cdot Width_{win} \leq x_b$</td>
</tr>
<tr>
<td></td>
<td>Bottom</td>
<td>$y_{win} + Scale \cdot Height_{win} \leq y_b - PH$</td>
</tr>
<tr>
<td></td>
<td>Left Side</td>
<td>$x_b + PH \leq x_{win}$</td>
</tr>
</tbody>
</table>

$(x_b, y_b) = \text{Print-point coordinates}$

$(x_{win}, y_{win}) = \text{Display window origin}$

### Table C-8

**Compressed Boundary Check Inequalities**

<table>
<thead>
<tr>
<th>Check</th>
<th>Inequality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top</td>
<td>$y_b + C_2 \leq y_{win}$</td>
</tr>
<tr>
<td></td>
<td>$x_{win} + Scale \cdot Width_{win} \leq x_b - C_1$</td>
</tr>
<tr>
<td>Bottom</td>
<td>$y_{win} + Scale \cdot Height_{win} \leq y_b - C_2$</td>
</tr>
<tr>
<td>Left Side</td>
<td>$x_b + C_3 \leq x_{win}$</td>
</tr>
</tbody>
</table>

$C_2 - C_1$ are defined in Table C-9
Table C-9

Boundary Check Constants

<table>
<thead>
<tr>
<th>( O_x )</th>
<th>( C_0 )</th>
<th>( C_1 )</th>
<th>( C_2 )</th>
<th>( C_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>PH</td>
<td>0</td>
<td>0</td>
<td>PW</td>
</tr>
<tr>
<td>1</td>
<td>PW</td>
<td>PH</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>PW</td>
<td>PH</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>PW</td>
<td>PH</td>
</tr>
</tbody>
</table>

If no boundary check proved true, the pattern is entirely or partially within the display window. The next step is to calculate the number of pattern rows and the starting row number that are within the display window. The "rows" are in respect to the base coordinate system's y-axis and not relative to the pattern storage definition. The algorithm must cover the four cases shown in Figure C-6 with the condition that the print-point may be on the "bottom" or "top" of the pattern box. The pattern's orientation must again be considered with the use of Equation C-2.

![Pattern Positions Within the Display Window](image)

**Figure C-6:** Pattern Positions Within the Display Window

The starting row within in a font pattern is always the row closest to the print-point that is still within the display area. This definition allows the display routines to access the pattern box data in the same manner regardless of orientation. But the calculation of the starting row and the number of rows will vary depending upon whether the pattern...
is oriented above or below a horizontal line drawn through the print-point of Figure C-5. The patterns with effective orientations below the horizontal line (0 and 1) need to be referenced by the x-axis of the display window coordinate system. The reason is that the data within the pattern box, which begins at the print-point, is ignored up to the starting row by the display routines. Likewise the patterns with orientations above the horizontal line will be referenced by the last row of the display window. The difference between the print-point and these reference lines (shown in Table C-10 as a function of the effective orientation) forms the basis for the row trimming algorithm given in Figure C-7.

In a similar manner the starting column and the number of columns within the display window are also calculated and placed in the filter structure. The "columns" are in respect to the base coordinate system's x-axis and not relative to the pattern storage definition. The number of columns outside the display window are calculated by the difference between the y-axis of the display window or the last column of the display window, depending upon the effective orientation of the pattern as shown in Table C-11. This time a vertical line drawn through the print-point separates those pattern orientations that are handled differently. The column trimming algorithm is shown in Figure C-8. The row and column trimming algorithms are very similar and may be combined to form one function.
<table>
<thead>
<tr>
<th>$O_s$</th>
<th>Rows</th>
<th>Rows Outside Display Window (extra)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>PH</td>
<td>$y_{win} - y_b$</td>
</tr>
<tr>
<td>1</td>
<td>PW</td>
<td>$y_{win} - y_b$</td>
</tr>
<tr>
<td>2</td>
<td>PH</td>
<td>$y_b - (y_{win} + \text{Scale} \times \text{Height}_{win} - 1)$</td>
</tr>
<tr>
<td>3</td>
<td>PW</td>
<td>$y_b - (y_{win} + \text{Scale} \times \text{Height}_{win} - 1)$</td>
</tr>
</tbody>
</table>

startrow = 0  
rows = Table C-10  
extra = Table C-10  
display_height = physical window_height (pixels) * viewing scale;

/* Note: Explanation uses "top" and "bottom" as a reference */  
/* for orientations 0 and 1. For other orientations "top" */  
/* and "bottom" are reversed. Only the comments change, */  
/* not the code. */  
/* If the top side of pattern is outside of display window: */  
if(extra > 0) {  
   /* Adjust the number of rows within the display window. */  
   rows -= extra;

   /* If the pattern continues through the bottom side of */  
   /* the display window then the pattern height is the */  
   /* size of the window. */  
   if(rows > display_height) rows = display_height;

   /* The number of rows above the top of the display */  
   /* window move the pattern's starting row. */  
   startrow = extra;
}  
else {  
   /* If the pattern's bottom edge exceeds the bottom of the */  
   /* display window boundary then trim off the bottom of */  
   /* the pattern. */  
   if(display_width + extra < cols) cols = display_width + extra;
}

Figure C-7: Row Trimming Algorithm
Table C-11
Col Trimming Algorithm Variable Initialization

<table>
<thead>
<tr>
<th>Os</th>
<th>Cols</th>
<th>Cols Outside Display Window (extra)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>PW</td>
<td>( x_{\text{win}} - x_p )</td>
</tr>
<tr>
<td>1</td>
<td>PH</td>
<td>( x_p - (x_{\text{win}} + \text{Scale} \times \text{Width}_{\text{win}} - 1) )</td>
</tr>
<tr>
<td>2</td>
<td>PW</td>
<td>( x_p - (x_{\text{win}} + \text{Scale} \times \text{Width}_{\text{win}} - 1) )</td>
</tr>
<tr>
<td>3</td>
<td>PH</td>
<td>( x_{\text{win}} - x_p )</td>
</tr>
</tbody>
</table>

startcol = 0
cols = Table C-11
extra = Table C-11
display_width = physical window_width (pixels) * viewing scale;

/* Note: Explanation uses "left" and "right" as a reference */
/* for orientations 0 and 3. For other orientations "left" */
/* and "right" are reversed. Only the comments change, not */
/* the code. */
/* If the left side of pattern is outside of display window: */
if(extra > 0) {
    /* Adjust the number of columns within the display window. */
    cols -= extra;

    /* If the pattern continues through the right side of the */
    /* display window then the pattern width is the size of */
    /* the window. */
    if(cols > display_width) cols = display_width;

    /* The number of columns to the left of the display */
    /* window move the starting column. */
    startcol = extra;
}
else {
    /* If the pattern’s right edge exceeds the right display */
    /* window boundary then trim off the pattern’s width. */
    if(display_width + extra < cols) cols = display_width + extra;
}

Figure C-8: Column Trimming Algorithm

C: Pattern Display 102
4) Size Filter

The size filter adjusts the values found by the area filter so that the starting row and column values and the number of rows and columns will begin on a physical screen pixel. The physical size of any given screen window, in pixels, remains constant. A scaling value may be applied to some windows to increase its logical size so that more of the document page may be viewed. The scale may be defined as the ratio of the logical display window size, in pels, to the physical display window size, in pixels. The scale is always constrained to be a positive integer value and is always applied to both the horizontal and vertical components of a window. So a window with a scale of 'n' only displays every n'th pel from every n'th row within the logical display window. If the scale is one, every pel within the logical display window maps to exactly one screen pixel and no adjustment of the filter values, calculated by the area filter, are necessary.

It is important to note that the origin of the physical and logical display window sizes on the document page always remains the same. When the scale is not one, a displayable font pattern’s starting row and column may not map to or align with a row and column within the logical page displayed on the graphics screen. This situation is depicted in Figure C-9 for a scale value of three. For the example in Figure C-9 the starting column does not need adjustment but the starting row value should be adjusted from row 0 to row 1 of the pattern box.
The adjustments to the starting row and columns depend upon the orientation of the pattern around the print-point. Again, Equation C-2 can be used to select proper equations. Table C-12 is used to calculate the starting row and column coordinates in the base coordinate system, since the starting row and column coordinates may no longer be the same as the print-point. To verify that the starting coordinates are a multiple of the window scale, the row and column remainders are calculated with the modulo operator in the following equations:
\[ R_{\text{col}} = (x_s - x_{\text{win}}) \times \text{scale} \]  
\[ R_{\text{row}} = (y_s - y_{\text{win}}) \times \text{scale} \]  
(Eq. C-4)  
(Eq. C-5)

<table>
<thead>
<tr>
<th>( O_s )</th>
<th>( x_s )</th>
<th>( y_s )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_b + \text{start col} )</td>
<td>( y_b + \text{start row} )</td>
</tr>
<tr>
<td>1</td>
<td>( x_b - \text{start col} )</td>
<td>( y_b + \text{start row} )</td>
</tr>
<tr>
<td>2</td>
<td>( x_b - \text{start col} )</td>
<td>( y_b - \text{start row} )</td>
</tr>
<tr>
<td>3</td>
<td>( x_b + \text{start col} )</td>
<td>( y_b - \text{start row} )</td>
</tr>
</tbody>
</table>

The row adjustment algorithm shown in Table C-13 can then be applied to prepare the filter structure information for the display routines. The column adjustment algorithms are identical to those in Table C-13 except that the effective orientations 3 and 1 must be interchanged and every word "row" should be replaced by "col". The adjustment algorithms also account for the possibility that the pattern may be within the logical window but the pattern box does not map to any physical window pixels.

<table>
<thead>
<tr>
<th>( O_s )</th>
<th>Adjustments</th>
</tr>
</thead>
</table>
| 0 & 3 | if\((R_{\text{row}} ! = 0)\) \{ 
\quad \text{startrow} += (\text{scale} - R_{\text{row}}); 
\quad \text{rows} -= (\text{scale} - R_{\text{row}}); 
\quad \text{if(rows} \leq 0) \text{skip display}; 
\} |
| 1 & 2 | \text{startrow} += R_{\text{row}}; 
\quad \text{rows} -= R_{\text{row}}; 
\quad \text{if(rows} \leq 0) \text{skip display}; |
5) Display the Pattern

The area and size filters determined whether a pattern was in the display window and provided information concerning how much of the pattern is within the display window. The display routines first recalculate the starting row and column in base coordinates according to Equation C-2 and Table C-12. The starting coordinates may then be translated into a physical screen location by the following equations:

\[
x_p = \frac{x_s - x_{win}}{scale} + x_{scrn} \quad \text{(Eq. C-6)}
\]

\[
y_p = \frac{y_s - y_{win}}{scale} + y_{scrn} \quad \text{(Eq. C-7)}
\]

where \((x_s, y_s)\) = Starting row/column coordinates from Table C-12.

\((x_{win}, y_{win})\) = Logical display window origin.

\((w_{scrn}, h_{scrn})\) = Physical display window screen origin.

The size filter ensures that the difference between the starting coordinates and the logical display window origin is always a multiple of the scale. To ensure that the display routines access the pattern data correctly, the "rows" and "columns" may need interchanging relative to the effective orientation. The redefinition of the "rows" and "columns" using Equation C-2 is shown in Table C-14. Finally, Table C-15 indicates how the screen pixel location must be incremented as the pattern data is sequentially fetched from memory for a given effective pattern orientation.
### Table C-14
Rcw & Column Redefinition

<table>
<thead>
<tr>
<th>Os</th>
<th>Cols (PW)</th>
<th>Rows (PH)</th>
<th>PW Offset</th>
<th>PH Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>cols</td>
<td>rows</td>
<td>start col</td>
<td>start row</td>
</tr>
<tr>
<td>1</td>
<td>rows</td>
<td>cols</td>
<td>start row</td>
<td>start col</td>
</tr>
<tr>
<td>2</td>
<td>cols</td>
<td>rows</td>
<td>start col</td>
<td>start row</td>
</tr>
<tr>
<td>3</td>
<td>rows</td>
<td>cols</td>
<td>start row</td>
<td>start col</td>
</tr>
</tbody>
</table>

### Table C-15
Plot Routine Pixel Location

<table>
<thead>
<tr>
<th>Os</th>
<th>Inner Loop (x)</th>
<th>Outer Loop (y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$x_p^{++}$</td>
<td>$y_p^{++}$</td>
</tr>
<tr>
<td>1</td>
<td>$y_p^{++}$</td>
<td>$x_p^{--}$</td>
</tr>
<tr>
<td>2</td>
<td>$x_p^{--}$</td>
<td>$y_p^{--}$</td>
</tr>
<tr>
<td>3</td>
<td>$y_p^{--}$</td>
<td>$x_p^{++}$</td>
</tr>
</tbody>
</table>

C: Pattern Display
6) End Point Determination

After a pattern has been printed, the PMP command set uses the cursor positioning parameters (1-27) to specify the next or end location of the cursor. By specifying an end point, another pattern may immediately be displayed without any separate or intervening move commands. Figure C-2 shows the end point in relation to the start and print-point for a font pattern. The discussion pertaining to finding the print-point is also applicable to finding the end point. The location of the end point also depends upon the current printing direction and on the pattern load orientation. Equations to calculate the end point coordinates for a particular print direction are shown in Table C-16. The equations in Table C-16 are only valid for a portrait pattern load orientation. Table C-17 shows the equations necessary to calculate the print-point location when the pattern load orientation is taken into account. The equations in Table C-17 are only valid for the "across" print direction.
### Table C-16
End Point Location as a Function of Print Direction

<table>
<thead>
<tr>
<th>Print Direction</th>
<th>$x_{end}$</th>
<th>$y_{end}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>across</td>
<td>$x + (LT + RB + PW)$</td>
<td>$y$</td>
</tr>
<tr>
<td>back</td>
<td>$x - (LT + RB + PW)$</td>
<td>$y$</td>
</tr>
<tr>
<td>down</td>
<td>$x$</td>
<td>$y + (LT + RB + PH)$</td>
</tr>
<tr>
<td>up</td>
<td>$x$</td>
<td>$y - (LT + RB + PH)$</td>
</tr>
</tbody>
</table>

### Table C-17
End Point as a Function of Load Orientation

<table>
<thead>
<tr>
<th>Load Orientation</th>
<th>$x_{end}$</th>
<th>$y_{end}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>$x + LT + RB + PW$</td>
<td>$y$</td>
</tr>
<tr>
<td>LR</td>
<td>$x + LT + RB + PH$</td>
<td>$y$</td>
</tr>
<tr>
<td>PU</td>
<td>$x + LT + RB + PW$</td>
<td>$y$</td>
</tr>
<tr>
<td>LL</td>
<td>$x + LT + RB + PH$</td>
<td>$y$</td>
</tr>
</tbody>
</table>
APPENDIX D
PAGE PREVIEWER OPERATION

The purpose of this appendix is to describe the operation of the page previewer's user interface. The SCRIPT documents "slide" and "schedule" are used as examples to demonstrate the capabilities of the previewer. These particular documents were chosen as examples because they sample various attributes of SCRIPT and they show the difference between a portrait and landscape document. These documents were not intended to examine all supported SCRIPT commands. Only a few features such as equations, vectors, and several font sizes are shown. The user interface to the PMP interpreter is covered in the following format:

- System Requirements 111
- Example Documents 112
  "Slide" Source SCRIPT 112
  "Slide" Printed Page (Actual Size) 114
  "Schedule" Source SCRIPT 115
  "Schedule" Printed Page (Actual Size) 122
- Previewing Options 123
  Changing Pages 126
  Full Page Display 127
  Scan Page Display 130
System Requirements

The page previewer software supports a personal computer (PC) machine class equivalent to the PS/2 model 30 as a minimum configuration. PC hardware requirements include an MCGA or VGA display, at least 1 megabyte of disk drive space, and 640 kilobytes of memory. In addition, a connection to an IBM mainframe must be available through a serial port. The page previewer requires DOS version 3.3 or higher and the HyperACCESS/5 [11] communication package. The IBM mainframe should support VM/XA CMS along with the document composition facility (SCRIPT). PC access to the mainframe should be available through a 7171 ASCII device attachment control unit (protocol converter).
This is font6
This is a test slide
We can use this method to generate slides for presentations
It should be easier than other methods
And a whole lot neater

This is font8
This is a test slide
We can use this method to generate slides for presentations
It should be easier than other methods
And a whole lot neater

This is font10
This is a test slide
We can use this method to generate slides for presentations
It should be easier than other methods
And a whole lot neater

This is font12
This is a test slide
We can use this method to generate slides for presentations
It should be easier than other methods
And a whole lot neater

This is font14
This is a test slide
We can use this method to generate slides for presentations
It should be easier than other methods
And a whole lot neater

D: Previewer Operation
This is font16
This is font16
We can use this method to generate slides for presentations.
It should be easier than other methods.
And a whole lot neater.

This is forteq=font12

.setsym smff
.se ask 'alpha sub k'
:df frame=none align=center.
s(x) = (i + sigma) u sub 0 - q over 1 %
    left lb 1 + delta b+2
    sum from <k=1> to infinity of
    <a sub k + b tanh &ask delta>
    over
    <&ask ( &ask sup 2 + b &ask
    tanh &ask delta + 1 ) > %
    cos &ask x right rb
:edf.
:df frame=none align=center.
<var delta sup 2 phi> over <var delta x sup 2> % + %
<var delta sup 2 phi> over <var delta y sup 2> % + %
<var delta sup 2 phi> over <var delta z sup 2> % = % 0
:edf.

D: Previewer Operation
\[ s(x) = (l + \sigma)u_0 - \frac{q}{I} \left[ 1 + \delta b + 2 \sum_{k=1}^{\infty} \frac{a_k + b \tanh \alpha_k \delta}{\alpha_k (\alpha_k^2 + b \alpha_k \tanh \alpha_k \delta + 1)} \right] \cos \alpha_k x \]
"Schedule" Source SCRIPT

.* filename is SHARBTEC GRIDMSTR
.* this is rotated 90 degrees by the font - no need to rotate w/ device
.* use SC fileid (d 3812)
.* do not erase this file

*** begin the variables for the BTEC newsletter session grid
.* key to variable names
.* se pp=ptxtttt where
.* pp = proj code. do=docc, ds=dsap, m=ofc mid-rg (old)
.* ow=owl (new 03-91 - was office mid-range)
.* mv=ofc nvs, vm=ofc vm
.* x = num of occurences of a sess for that proj in that day/time
.* tttt = time. 8:00 a.m. = 0800 ... 4:30 p.m. = 0430
.* d = day of the week. Sun=a Mon=b Tue=c Wed=d Thu=e Fri=f
.* sort by time ( 8-11 ) day (12 -12 ) proj (5-6) proj occur (7-7)
.* sort (to end of session ONLY) 8 12 5 7
.* begin sort on next line
.* .5....1.+....2.+....3.+....4.+....5.+....6.+....7...
.* 'regular session variables begin here
.se dol0800c '1770 Host Publishing Product Ovw
.se ds10800c '1955 Relat’l & AS I/f to L.l-2-3/M
.se mv10800c '1910 OV/MVS E’prz Addr Bk Ovw
.se mv20800c '1911 OV/MVS Sys Lvl Cap Plan & Perf
.se ow10800c '1810 OV/2 LAN R2 Demo - Cal
.se ow20800c '1817 P-M807 ... When You Need a LAN
.se vm10800c '1692 OV/VM Arch. Ovw.
.se bt10800d '1658 Open Systems, IBM, & You
.se do10800d '1782 P-1055 IBM GRAPHIGS
.se ds10800d '1961 Tuning MVS Appl Sys Perf
.se mv10800d '1920 U Expr: Appl I/g - Appl Conn
.se mv20800d '1921 OV/MVS R2 Early Instal Pgm
.se mv30800d '1941 Primary session is I658
.se vm10800d '1702 Fin. Just. of Info Delv Sy
.se vm20800d '1710 Primary session is I658
.se bt10800e '1665 IBM’ s Intrl OfcVn Implem
.se dol0800e '1761 Managing Documents
.se ds10800e '1967 Pract’l Appl for Adv’d Techno
.se mv10800e '1930 Primary session is I631
.se ow10800e '1830 IBM Standalone - Updtd/Demo
.se ow20800e '1831 OV/MVS R 2 DOs Ofc Dir Conn
.se vm10800e '1707 Primary session is I665
.se vm20800e '1727 PROPS In Your Pocket
.se vm30800e '1737 Primary session ia I831
.se mv10800f '1940 Dr Rufe’s Rx for Safe O/MVS
.se ow10800f '1840 OWL Rqmnts & SHARE 77 Plan
.se vm10800f '1746 OV/VM Lrg Sys’s Rnd Tbl
.se do10930c '1757 GDDM Gx for APP
.se do20930c '1771 Desktop Publishing Overview
.se ds10930c '1969 Appl Sys /PAS I/g
.se mv10930c '1912 Mgr to OV/MVS E’prz Addr Bk
.se ow10930c '1811 OV/2 LAN R2 Demo - Dir’ty
.se se10930c '1694 OV Customer Educ
.se se20930c '1695 OV/VM Wh. Pap. Forum
.se do10930d '1754 APP Perf - Prt w sp of light
.se ds10930d '1962 AS Proj Mgmt Rqmnts
.se mv10930d '1922 OV/MVS R2 Erly Instal Pgm
.se mv20930d '1947 Primary session is I819
.se ow10930d '1819 Ofc Wkrsta. Connectivity FFA
.se vm10930d '1703 OfficeVision Horizons
.se ve20930d '1926 P-1819 Ofc W’sta Conn FFA
.se bt10930e '1665 IBM Info N’w’rk Expedite Mail
.se bc20930e '1667 Gartner Grp - on Prt Arch
.se do10930e '1794 Ex’s of Sub-Surface Imaging
.se ds10930e '1968 Executive Decsn/MVS

D: Previewer Operation
* end of 8:00 a.m. time sessions
* beginning of sessions for 9:30 a.m. time slot

~9:30 - BTEC ~ rbl. ~ & rbl. ~ & rbl. ~ & btl10930.e ~ & rbl. 
  ~ BTEC ~ rbl. ~ & rbl. ~ & rbl. ~ & btl20930.e ~ & rbl. 
  ~ DOCC ~ rbl. ~ & dol10930.c ~ & dol10930.d ~ & dol10930.e ~ & dol10930.f 
  ~ DOCC ~ spec8 ~ & dol20930.c ~ & rbl. ~ & rbl. 
  ~ DSSP ~ spec2 ~ & dll10930.c ~ & dll10930.d ~ & dll10930.e ~ & rbl. 
  ~ OMV ~ spec3 ~ & mvl10930.c ~ & mvl10930.d ~ & mvl10930.e ~ & mvl10930.f 
  ~ OMV ~ spec4 ~ & rbl. ~ & mvl20930.c ~ & mvl20930.e ~ & rbl. 
  ~ OMV ~ spec5 ~ & rbl. ~ & mvl30930.e ~ & rbl. 
  ~ OVM ~ spec6 ~ & vml10930.c ~ & vml10930.d ~ & vml10930.e ~ & vml10930.f 
  ~ OVM ~ spec7 ~ & vml20930.c ~ & vml20930.d ~ & vml20930.e ~ & rbl. 
  ~ OVM ~ rbl. ~ & rbl. ~ & vml30930.e ~ & rbl. 
  ~ OWL ~ rbl. ~ & rbl. ~ & vml930.e ~ & rbl. 
  ~ OWL ~ rbl. ~ & rbl. ~ & rbl. ~ & rbl. 

* end of 9:30 a.m. time sessions
* beginning of sessions for new time period

~11:00 - BTEC ~ rbl. ~ & rbl. ~ & btl11100.d ~ & btl11100.e ~ & rbl. 
  ~ DOCC ~ dol1100.b ~ & dol1100.c ~ & dol1100.e ~ & dol1100.f 
  ~ DOCC ~ rbl. ~ & dol21100.c ~ & dol21100.d ~ & dol21100.e ~ & dol21100.f 
  ~ DSSP ~ dol1100.b ~ & dol1100.c ~ & dol1100.d ~ & dol1100.e ~ & dol1100.f 
  ~ OMV ~ mvl1100.b ~ & mvl1100.c ~ & mvl1100.d ~ & mvl1100.e ~ & mvl1100.f 
  ~ OMV ~ rbl. ~ & mvl21100.d ~ & rbl. ~ & rbl. 
  ~ OVM ~ vml1100.b ~ & vml1100.c ~ & vml1100.d ~ & vml1100.e ~ & vml1100.f 
  ~ OVM ~ rbl. ~ & vml21100.d ~ & rbl. ~ & rbl. 
  ~ OWL ~ sow11100.b ~ & sow11100.d ~ & sow11100.e ~ & sow11100.f 
  ~ OWL ~ sow11100.b ~ & sow11100.d ~ & sow11100.e ~ & rbl. 

* end of sessions for 11:00 a.m. time slot
* beginning of sessions for 1:30 p.m. time slot

~1:30 - BTEC ~ & btl11330.b ~ & rbl. ~ & rbl. ~ & rbl. 
  ~ DOCC ~ dol11330.c ~ & dol11330.d ~ & dol11330.e ~ & dol11330.f 
  ~ DOCC ~ rbl. ~ & dol211330.d ~ & dol211330.e ~ & dol211330.f 
  ~ DSSP ~ dol11330.b ~ & dol11330.c ~ & dol11330.d ~ & dol11330.e ~ & dol11330.f 
  ~ OMV ~ mvl11330.b ~ & mvl11330.c ~ & mvl11330.d ~ & mvl11330.e ~ & rbl. 
  ~ OMV ~ rbl. ~ & mvl211330.d ~ & rbl. ~ & rbl. 
  ~ OVM ~ vml11330.b ~ & vml11330.c ~ & vml11330.d ~ & vml11330.e ~ & vml11330.f 
  ~ OWL ~ sow11330.b ~ & sow11330.c ~ & sow11330.d ~ & sow11330.e ~ & rbl. 

* end of sessions for 1:30 p.m. time slot
* beginning of sessions for 3:00 p.m. time slot

~3:00 - BTEC ~ rbl. ~ & btl11500.d ~ & rbl. 
  ~ DOCC ~ dol11500.b ~ & dol11500.c ~ & dol11500.d ~ & dol11500.e ~ & rbl. 
  ~ DOCC ~ rbl. ~ & dol211500.d ~ & dol211500.e ~ & rbl. 
  ~ DSSP ~ dol11500.b ~ & dol11500.c ~ & dol11500.d ~ & dol11500.e ~ & rbl. 
  ~ OMV ~ vml11500.b ~ & vml11500.c ~ & vml11500.d ~ & vml11500.e ~ & rbl. 
  ~ OWL ~ sow11500.b ~ & sow11500.c ~ & sow11500.d ~ & sow11500.e ~ & rbl. 

* end of sessions for 3:00 p.m. time slot
* beginning of sessions for 4:30 p.m. time slot

~4:30 - BTEC ~ & btl11630.c ~ & btl11630.d ~ & rbl. 
  ~ DOCC ~ dol11630.b ~ & dol11630.c ~ & dol11630.d ~ & dol11630.e ~ & rbl. 
  ~ DOCC ~ rbl. ~ & dol211630.d ~ & rbl. 
  ~ DSSP ~ rbl. ~ & rbl. ~ & rbl. ~ & rbl. 

D: Previewer Operation 119
- OMVS- &mv11630b.-&mv11630c.-&mv11630d.-&mv11630e.-&rbl.
- OMVS- &mv21630b.-&rbl. -&mv21630d.-&mv21630e.-&rbl.
- OMVS- &rbl. -&rbl. -&mv31630d.-&rbl.
- OVM -&rbl. -&mv11630c.-&mv11630d.-&mv11630e.-&rbl.
- OVM -&rbl.-&mv11630c.-&mv11630d.-&mv11630e.-&rbl.
- OWL -&owl11630b.-&owl11630c.-&owl11630d.-&owl11630e.-&rbl.
- OWL -&owl21630b.-&owl21630c.-&owl21630d.-&rbl.
- OWL -&owl31630d.-&owl11630b.-&owl11630c.-&owl11630d.-&owl11630e.-&rbl.

* end of sessions for 4:30 p.m. time slot

.. beginning of sessions for 6:00 p.m. time slot

-6:00 -BTEC- &bcl11800b.-&rbl. -&rbl. -&rbl. -&rbl.
- DOCC- &dcl11800b.-&rbl. -&rbl. -&rbl. -&rbl.
- OMVS- &mv11800b.-&rbl. -&rbl. -&rbl. -&rbl. -&rbl.
- OMVS- &mv11800d.-&rbl. -&rbl. -&rbl. -&rbl. -&rbl.
- OVM -&rbl. -&mv11800c.-&rbl. -&rbl. -&rbl. -&rbl.
- OVM -&rbl. -&mv21800c.-&rbl. -&rbl. -&rbl. -&rbl.
- OWL -&owl11800b.-&owl11800c.-&owl11800d.-&owl11800e.-&owl11800d.-&owl11800e.-&rbl.

* end of sessions for 6:00 p.m. time slot

.. beginning of sessions for 7:30 p.m. time slot

-7:30 -DOCC- &dcl11930b.-&rbl. -&rbl. -&rbl. -&rbl.

.. off

* new code begins here for abbrev key
.ar off
.ar key on
.bat key
* begin the session grid key here
*BTEC Grid Key
* BTEC Grid Key
.ti - 05
tp 3mm 16mm
.bx 1mm 45mm
*KDY
.bx 1mm 14mm 45mm
* \AS - Application Sys
.Anncc - announcements
.*-Capb - capability
.Capc - capacity
.Cmmtc - committee
.Comm - communication
.Conn - connectivity
.*-Chat - 
.*-D - data
.Decsn - decision
.*-DISO - DISOSS
.DW - DisplayWrite, etc.
.Env - environment
.E'prz - enterprise
.*-ESP - early support program
.*-Ex - executive
.Expr - experience
.FFA - free-for-all
* -Futr - future
.Gx - 
 Graphics
.Hardw - hardware
.I/S - information systems
.I/a - 
Interactive, etc.
.I/c - interchange
.I/f - interface, interfacing
.I/g,Int/g - integrate, integrating
.Intrl - internal
.P- - primary session is...
.*-Netwk - network
.Offc - office
.OV - OfficeVision

D: Previewer Operation 120
-Ovw - overview
-**-Pers - personal
-**-P&I - planning for and installing
-Prt - printer
.* ~PRF - PROFS
-Rqmnts - requirements
.*-Rndtbl - roundtable
~Softw - software
~s.w. - software
.*-Sup - support
~Technl - technical
~Technq - technique
~Techno - technology
~Updt, Up - update
~U - user
~W'sta - workstation
.* ~WPap - white Paper
.bx off
.ar off
<table>
<thead>
<tr>
<th>TIME</th>
<th>MONDAY</th>
<th>TUESDAY</th>
<th>WEDNESDAY</th>
<th>THURSDAY</th>
<th>FRIDAY</th>
</tr>
</thead>
<tbody>
<tr>
<td>9:00</td>
<td>BTEC</td>
<td>Host Publishing Product Owning</td>
<td>IBM Systems/IBM, &amp; You</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>Managing Documents</td>
</tr>
<tr>
<td></td>
<td>BTEC</td>
<td>Monday</td>
<td>IBM &amp; You</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
</tr>
<tr>
<td></td>
<td>DOCO</td>
<td>Project Sessions</td>
<td>IBM &amp; You</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
</tr>
<tr>
<td></td>
<td>DSSP</td>
<td></td>
<td>IBM &amp; You</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
</tr>
<tr>
<td></td>
<td>OMVS</td>
<td></td>
<td>IBM &amp; You</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
</tr>
<tr>
<td></td>
<td>General Session at</td>
<td></td>
<td>IBM &amp; You</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
</tr>
<tr>
<td></td>
<td>9:00 a.m.</td>
<td></td>
<td>IBM &amp; You</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
</tr>
<tr>
<td></td>
<td>OVM</td>
<td></td>
<td>IBM &amp; You</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
</tr>
<tr>
<td></td>
<td>OWL</td>
<td></td>
<td>IBM &amp; You</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IBM &amp; You</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IBM &amp; You</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IBM &amp; You</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IBM &amp; You</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
</tr>
<tr>
<td>11:00</td>
<td>BTEC</td>
<td>DOCC</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
</tr>
<tr>
<td></td>
<td>DOCO</td>
<td>Project Open, Prod. Surv</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
</tr>
<tr>
<td></td>
<td>DSSP</td>
<td></td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
</tr>
<tr>
<td></td>
<td>OMVS</td>
<td></td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
</tr>
<tr>
<td></td>
<td>General Session at</td>
<td></td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
</tr>
<tr>
<td></td>
<td>9:00 a.m.</td>
<td></td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
</tr>
<tr>
<td></td>
<td>OVM</td>
<td></td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
</tr>
<tr>
<td></td>
<td>OWL</td>
<td></td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
</tr>
<tr>
<td>13:30</td>
<td>BTEC</td>
<td>DOCC</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
</tr>
<tr>
<td></td>
<td>DOCO</td>
<td>Project Open, Prod. Surv</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
</tr>
<tr>
<td></td>
<td>DSSP</td>
<td></td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
</tr>
<tr>
<td></td>
<td>OMVS</td>
<td></td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
</tr>
<tr>
<td></td>
<td>General Session at</td>
<td></td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
</tr>
<tr>
<td></td>
<td>9:00 a.m.</td>
<td></td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
</tr>
<tr>
<td></td>
<td>OVM</td>
<td></td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
</tr>
<tr>
<td></td>
<td>OWL</td>
<td></td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
</tr>
<tr>
<td>3:00</td>
<td>BTEC</td>
<td>DOCC</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
</tr>
<tr>
<td></td>
<td>DOCO</td>
<td>Project Open, Prod. Surv</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
</tr>
<tr>
<td></td>
<td>DSSP</td>
<td></td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
</tr>
<tr>
<td></td>
<td>OMVS</td>
<td></td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
</tr>
<tr>
<td></td>
<td>General Session at</td>
<td></td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
</tr>
<tr>
<td></td>
<td>9:00 a.m.</td>
<td></td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
</tr>
<tr>
<td></td>
<td>OVM</td>
<td></td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
</tr>
<tr>
<td></td>
<td>OWL</td>
<td></td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
<td>IBM &amp; Inter Office Implant SoE</td>
</tr>
</tbody>
</table>

**Time:** 4:30 p.m. to 7:00 p.m.

**Location:** 1792 Penguin Mixer
Previewing Options

The previewing interface begins by displaying page 1 of the SCRIPT document as shown in the screen dump of Figure D-3. A menu bar is shown at the top of the screen and is divided into four sections. The sections, starting on the far right, control options for paging, full page display, scan page display, and exiting the previewer. Windows that always display the full page of a document are referred to as full page windows. Full page windows are used to give the overall format of the page and are further used to define scan page views. Scan page windows selectively display that portion of a page defined by a frame located within the full page window. Scan page windows allow the user to "zoom" in and view details in a "what you see is what you get" (WYSIWYG) format. Figure D-3 shows a full page window with the scan page frame located in the lower center of the page. Figure D-4 shows the scan page window corresponding to Figure D-3. Previewing document pages through manipulation of full and scan page windows will be explained as the menu options are presented.

Selection of the menu options may be accomplished with a mouse or keyboard. To select an option with the mouse, move the cursor to the option box and press the left mouse button. The first letter of each option box title may also be entered at the keyboard to select a menu option. For example, the option which terminates the current previewing session may be selected by clicking within the "EXIT" box with the mouse or pressing "E" from the keyboard. The menu groups that control changing pages and manipulating the full and scan page windows will be presented in the following sections.
\[
\begin{bmatrix}
1 + \delta b + 2 \\
\sum_{k=1}^{8} \frac{a_k + b \tanh \alpha}{\alpha_k (\alpha_k^2 + b \alpha_k \tanh \alpha)}
\end{bmatrix}
\]

\[
\begin{align*}
\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} + \frac{\partial^2 \phi}{\partial z^2} &= 0 
\end{align*}
\]

Figure D-4: Scan Page Window of Figure D-3
Changing Pages

The menu section in the upper right corner of the screen controls paging options. The displayed page's number along with the total number of pages in the document is shown on the menu. To move forward through a document select the "+" menu option. The keyboard input "+" or "PgDn" will also result in the next page being displayed. Similarly, selecting the "-" option with the mouse or with the "-" or "PgUp" keys will cause the previous page to be displayed. The pages are numbered consecutively, beginning at zero, and do not coincide with the page numbers that SCRIPT assigns. Page zero is a cover page indicating the date, time, and origin of the document. Selecting the "+" option on the last page, or the "-" option on page zero, will reset the page display to the beginning or end of the document respectively.
Full Page Display

Initially, pages are displayed within a full page window in a portrait orientation. A portrait orientation has an aspect ratio such that the page width is less than the page height. The previewer also supports landscape orientations in which the page width is greater than the page height. The orientation of a full page window may be changed with the "ORIENTATION" menu option. Selecting the "ORIENTATION" option switches the full page window from a portrait to a landscape view and vice versa by rotating the page 90 degrees. Figure D-5 shows the "schedule" document after using the "ORIENTATION" option. The scan page window for Figure D-5 is shown in Figure D-6.

The 3812 page printer, which the page previewer emulates, can produce pages and patterns in any 90 degree increment. The "ORIENTATION" option will rotate a page into any 90 degree orientation. The "ROT 180" option rotates a page to an upside-down position while remaining in portrait or landscape mode.
<table>
<thead>
<tr>
<th>Time</th>
<th>Session Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wed</td>
<td>I658 Open Systems, IBM, &amp; You</td>
</tr>
<tr>
<td></td>
<td>I782 P-I055 IBM GRAPHICS</td>
</tr>
<tr>
<td></td>
<td>I665 IBM's Interns</td>
</tr>
<tr>
<td></td>
<td>I761 Managing</td>
</tr>
<tr>
<td></td>
<td>I961 Tuning MVS Appl Sys Perf</td>
</tr>
<tr>
<td></td>
<td>I920 U Expr: Appl I/g - Appl Conn</td>
</tr>
<tr>
<td></td>
<td>I921 OV/MVS R2 Early Instal Pgm</td>
</tr>
<tr>
<td></td>
<td>I941 Primary session is I658</td>
</tr>
<tr>
<td></td>
<td>I702 Fin. Just. of Info Delv Sy</td>
</tr>
<tr>
<td></td>
<td>I710 Primary session is I658</td>
</tr>
<tr>
<td></td>
<td>I707 Primary session</td>
</tr>
<tr>
<td></td>
<td>I727 PROFS In</td>
</tr>
<tr>
<td></td>
<td>I737 Primary session</td>
</tr>
<tr>
<td></td>
<td>I830 IBM Standards</td>
</tr>
<tr>
<td></td>
<td>I831 OV/MVS</td>
</tr>
<tr>
<td></td>
<td>I666 IBM Information</td>
</tr>
<tr>
<td></td>
<td>I667 Gartner G.</td>
</tr>
<tr>
<td></td>
<td>I794 Ex's of S</td>
</tr>
<tr>
<td></td>
<td>I754 AFP Perf- Prt w sp of light</td>
</tr>
</tbody>
</table>

*Figure D-6: Scan Page Window of Figure D-5*
Scan Page Display

The scan page displays the area within the scan page frame shown on the full page window. The scan page and full page may be alternately displayed by clicking the "right" mouse button or by pressing the carriage return key. While viewing a full page window the scan page frame may be moved by locating the mouse cursor in the upper left hand corner of the area to which the frame should enclose and clicking the "left" mouse button. The numeric keypad arrows also move the scan page frame. Similarly, panning may be accomplished within a scan page window by clicking the "left" mouse button which will move the area under the mouse cursor to the center of the screen. The numeric keypad arrows also allow panning within scan page windows.

The previewer supports a split screen mode in which portrait full page windows may be displayed simultaneously with the scan page. Selecting the "VIEW" menu option toggles the scan page between split and full screen views. The split screen mode is shown in Figure D-7. Panning within the scan page window of a split screen is not allowed since the scan page frame is available.

The scan page may be scaled to view a larger area of a page. The menu bar displays the scale of the scan page. A scale of 1 indicates that every printable pel is being displayed at the resolution of the 3812 page printer (240 pels/inch). A scale of 2 doubles the width and length of the scan page frame which quadruples the area shown within a scan page. Increasing the scale reduces pattern resolution but increases the area that may be viewed. Figures D-8 and D-9 show a portion of the "slide" document with a scale of 2 and 3 respectively. These figures may be compared with Figure D-4 that has a scale of 1. The scan page
\[ \sum_{k=1}^{\infty} \frac{a_k}{\alpha_k (\alpha_k^2 + 1)} + 2 \sum_{k=1}^{\infty} \frac{a_k}{\alpha_k (\alpha_k^2 + 1)} \]

\[ \frac{\partial^2 \phi}{\partial y^2} + \frac{\partial^2 \phi}{\partial z^2} \]

Figure D-7: Split Screen Mode
frame does not leave the boundaries of a full page window and so the scale is limited to a maximum of 4. The scale is always an integer and may be increased by clicking within the "SCALE" menu box or by pressing "S" from the keyboard. Attempting to increase the scale beyond the maximum will result in the scale returning to 1. The numbers 1 through 4 may also be pressed at the keyboard to change the scale directly.
This is a test slide
We can use this method to generate slides for presentations
It should be easier than other methods
And a whole lot neater

This is forteq = font12

\[ 1 = (l + \sigma)u_0 - \frac{q}{l} \left[ 1 + \delta b + 2 \sum_{k=1}^{\infty} \frac{u_k + b \tanh \alpha_k \delta}{x_k(x_k^2 + b\alpha_k \tanh \alpha_k \delta + 1)} \cos \alpha_k x \right] \]

\[ \frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} + \frac{\partial^2 \phi}{\partial z^2} = 0 \]

\textbf{Figure D-8:} Scan Page Example (Scale = 2)
It should be easier than other methods
And a whole lot neater
This is font14
This is a test slide
We can use this method to generate
slides for presentations
It should be easier than other methods
And a whole lot neater
This is font16
This is a test slide
We can use this method to generate
slides for presentations
It should be easier than other methods
And a whole lot neater

This is font12

\[ s(x) = (l + \sigma)u_0 - \frac{y}{l} \left[ 1 + \delta b + 2 \sum_{k=1}^{\infty} \frac{u_k + b \tanh \alpha_k \delta}{\alpha_k (\alpha_k^2 + b \alpha_k \tanh \alpha_k \delta + 1)} \cos \alpha_k \right] \]

\[ \frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} + \frac{\partial^2 \phi}{\partial z^2} = 0 \]

Figure D-9: Scan Page Example (Scale = 3)
VITA

Stephen Gibbs Bears was born on February 14, 1965 in Fairport, New York and was raised in Mount Vernon, Virginia. He graduated from Virginia Polytechnic Institute and State University with a B.S. degree in Mechanical Engineering in 1988. As an undergraduate he worked for Belvoir Research Development & Engineering Center as an engineering cooperative education student. Following graduation he reentered Virginia Tech to pursue an M.S. degree in Electrical Engineering. After graduation from Virginia Tech he plans to seek work that combines both his interest in Computer Engineering and Mechanical Engineering.