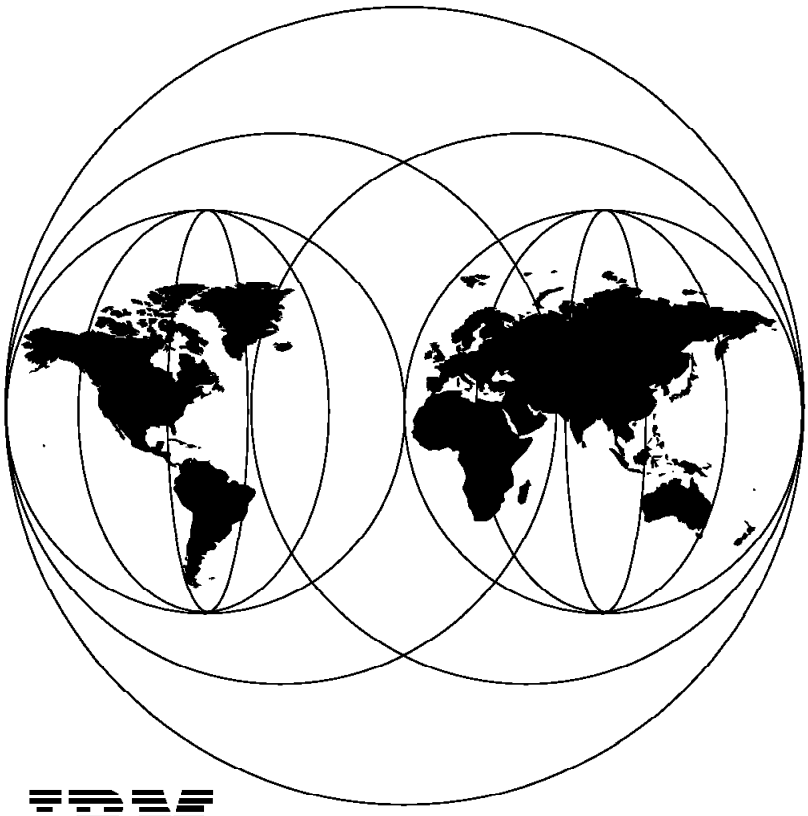
International Technical Support Organization

SG24-4523-00

Plumbers' Workbench
CMS Pipelines and OS/2 Pipe Synergy

December 1995



IBM

International Technical Support Organization
Poughkeepsie Center

International Technical Support Organization

SG24-4523-00

Plumbers' Workbench
CMS Pipelines and OS/2 Pipe Synergy

December 1995

Take Note!

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

First Edition (December 1995)

This edition applies to Version 1 of the Plumbers’ Workbench, for use with the VM/Enterprise Systems Architecture and the OS/2 operating systems.

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

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

IBM Corporation, International Technical Support Organization
Dept. 541 Mail Station P099
522 South Road
Poughkeepsie, New York 12601-5400

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

Abstract

The purpose of this document is to provide solutions for the following problems:

VM/ESA is rich in its client/server APIs and products; however, when simple seamless program-to-program communication between a workstation and host is sought, one is most often left with a tool that is either so low level that it requires a large programming investment, or so high level that it suffers from inflexibility.

CMS Pipelines is unquestionably a programmer productivity booster, but multistream pipes may be difficult to conceptualize. Add to this the variableless nature of a pipe and the common use of minimal labels, and the occasional *CMS Pipelines* user may require a visual framework in order to plumb like the pros.

The Plumbers' Workbench solves these problems. Phase I, as shipped with this document on the included diskette, offers a seamless environment that spans OS/2 and CMS. OS/2 pipes are extended to interact with *CMS Pipelines*, combining the strengths of both environments. Phase II, a future project, will integrate a graphical environment with Phase I to display and manage the pipeline topology with a point-and-click GUI.

This document was written for CMS users and programmers who wish to exploit the synergy of OS/2 and CMS through a unified pipeline client/server environment. Some knowledge of OS/2 and CMS is assumed.

(91 pages plus index.)

Contents

Figures

Special Notices

This publication is intended to help CMS users to use the Plumbers' Workbench. The information in this publication is not intended as the specification of any programming interfaces that are provided by OS/2 and VM/Enterprise Systems Architecture. See the PUBLICATIONS section of the IBM Programming Announcement for OS/2 and VM/Enterprise Systems Architecture for more information about what publications are considered to be product documentation.

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

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

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

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

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

AIX	BookMaster
C/370	CUA
IBM	OS/2
Presentation Manager	Virtual Machine/Enterprise Systems Architecture
VM/ESA	WIN-OS/2

The following terms are trademarks of other companies:

C-bus is a trademark of Corollary, Inc.

POSIX is a trademark of the Institute of Electrical and Electronic Engineers.

Network File System is a trademark of Sun Microsystems, Incorporated.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

PostScript is a trademark of Adobe Systems Incorporated.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Windows is a trademark of Microsoft Corporation.

Other trademarks are trademarks of their respective companies.

Preface

This document serves these purposes:

- It describes the Plumbers' Workbench as a new integrated environment for creation and testing of *CMS Pipelines* applications. Thus, it explains how to configure the Plumbers' Workbench and how to make best use of its features; it shows how to use the Plumbers' Workbench.
- It is intended to help programmers who are implementing a distributed client/server application on CMS and OS/2. It does this by describing how the Plumbers' Workbench was designed and implemented.

How This Document Is Organized

The document is organized as follows:

- Chapter 1, "Introduction" provides an overview of the Plumbers' Workbench project and its deliverables.

Part 1, "User's Guide"

- Chapter 2, "Installation Planning" describes some decisions you must make before you can install the Plumbers' Workbench.
- Chapter 3, "Installation" explains how to install the Plumbers' Workbench.
- Chapter 4, "Customization" describes how you insert your installation's parameters into the Plumbers' Workbench.
- Chapter 6, "VMPIPE—The Command Interface for the Plumbers' Workbench" is the reference for the command you use on the workstation to send requests to the Plumbers' Workbench server.
- Chapter 7, "Security" describes the security aspects of the Plumbers' Workbench.
- Chapter 8, "Messages" contains the message reference for the Plumbers' Workbench. It contains both client and server messages.

Part 2, "Implementation"

- Chapter 9, "The Design of the Plumbers' Workbench" explains the design of the Plumbers' Workbench.
- Chapter 10, "Development and Testing Environment" describes the configuration of the system we used to build the Plumbers' Workbench.
- Chapter 11, "Books We Have Read" contains a review of some books we read in the process of making the Plumbers' Workbench.
- Chapter 12, "Tips and Advice on Developing Client/Server Applications for OS2 and CMS" may be a useful source of information for the CMS programmer who is setting up shop on OS/2.
- Chapter 13, "Managing Documentation, Help, and Message Repositories" describes how to set up a common message repository for a distributed application.

Appendixes

- Appendix A, “REXX External Functions” documents the external REXX functions that the Plumbers’ Workbench uses to implement the base protocol layer.
- Appendix B, “Protocols” contains an overview of the base layer protocol.
- Appendix C, “Packet Formats” documents the individual packets that are used to implement the protocol stacks.
- Appendix D, “Program Directory” contains the list of programs and parts shipped with this publication.

Related Publications

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

- *VM/ESA: Connectivity Planning, Administration, and Operation*, SC24-5756
- *VM/ESA: CMS Pipelines Reference*, SC24-5778
- *VM/ESA: CMS Pipelines User's Guide*, SC24-5777

International Technical Support Organization Publications

- *GUI Facility Developer's Guide Using C++* , SG24-2542
- *Client/Server Computing with VM/ESA as Part of the Open Enterprise*, GG24-3950

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

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

To get a catalog of ITSO redbooks, VNET users may type:

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

A listing of all redbooks, sorted by category, may also be found on MKTTOOLS as ITSOPUB TXT. This package is updated monthly.

How to Order ITSO Redbooks

IBM employees in the USA may order ITSO books and CD-ROMs using PUBORDER. Customers in the USA may order by calling 1-800-879-2755 or by faxing 1-800-445-9269. Almost all major credit cards are accepted. Outside the USA, customers should contact their local IBM office.

Customers may order hardcopy ITSO books individually or in customized sets, called BOFs, which relate to specific functions of interest. IBM employees and customers may also order ITSO books in online format on CD-ROM collections, which contain redbooks on a variety of products.

ITSO Redbooks on the World Wide Web (WWW)

Internet users may find information about redbooks on the ITSO World Wide Web home page. To access the ITSO Web pages, point your Web browser (such as WebExplorer from the OS/2 3.0 Warp BonusPak) to the following:

```
http://www.redbooks.ibm.com/redbooks
```

IBM employees may access LIST3820s of redbooks as well. Point your web browser to the IBM Redbooks home page:

```
http://w3.itsc.pok.ibm.com/redbooks/redbooks.html
```

Getting the Plumbers' Workbench from the Net

The deliverables for the Plumbers' Workbench are available through the Internet and the IBM VNET network as a self-extracting zip file.

Internet: If you have an Internet connection, you can access the program materials by anonymous FTP:

```
ftp ftp.almaden.ibm.com
user: anonymous
password: your E-mail address
dir
cd redbooks/SG244523
binary
mget *.*
quit
```

Review the README.1ST file downloaded with the above commands for additional information.

VNET: Users of the IBM VNET network may get the SG244523 PACKAGE, which is stored in the VMTOOLS repository. Issue the following command from a CMS session:

```
TOOLS TO VMTOOLS GET SG244523 PACKAGE
```

Acknowledgments

This publication is the result of a residency conducted at the International Technical Support Organization, Poughkeepsie Center during the summer of 1995.

The author of this document is:

John P. Hartmann IBM Denmark

The residency was coordinated by:

Scott B. Vetter IBM ITSO, Poughkeepsie

We are grateful to Melinda W. Varian of Princeton University for test piloting the Plumbers' Workbench and for invaluable advice and guidance while we wrote this document.

Chapter 1. Introduction

This document is about the Plumbers' Workbench, a work still in progress. It is about how to use the Plumbers' Workbench and about how the Plumbers' Workbench was made.

The Plumbers' Workbench is an application that runs distributed between a user's workstation (the *client*) and the user's host session (the *server*).

The goal of the Plumbers' Workbench project is to build tools that will make users more productive through the synergy of OS/2 and CMS:

- A cross-platform pipeline pipes data directly between OS/2 programs and *CMS Pipelines* filters. This gives the OS/2 programmer easy access to the superior utilities of *CMS Pipelines*.
- A graphical environment for developing applications based on *CMS Pipelines* allows pipeline programmers to concentrate on the individual stages of a pipeline and their interconnections, thus increasing productivity. Because each stage will be specified individually, there will be no special characters and no labels to worry about. The topology of a multistream pipeline specification will be directly visible; the pipeline architect will be able to connect streams by "point-and-click".
- A graphical environment for showing how data flows in a pipeline. This will visualize the data flow and thus make multistream pipelines more approachable by the "journeyman plumber".

Plumbers' Workbench function is implemented through the protocol stacks shown in Figure 1.

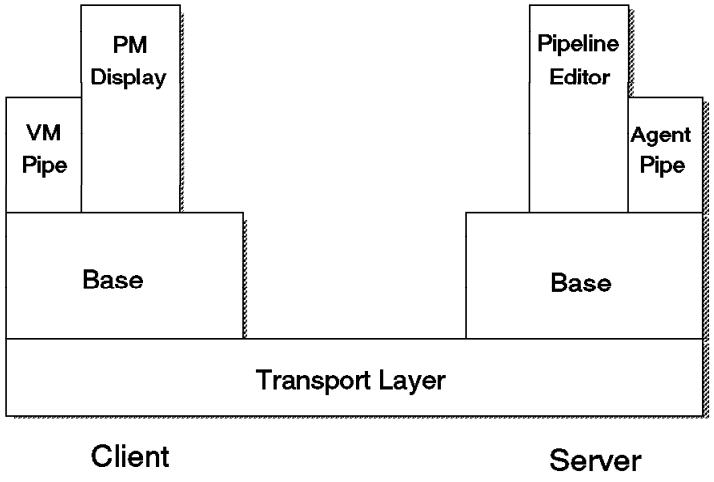


Figure 1. Components of the Plumbers' Workbench

On the client side, OS/2 pipe data is read by a new OS/2 program, VMPIPE, and passed to the agent on the server side by a transport, such as TCP/IP. The agent connects the OS/2 standard input to the input stream for the CMS Pipeline which is entered as the argument of the VMPIPE command. The results are

returned to the client and the transformed data flow from the VMPIPE command through the remainder of the OS/2 pipe.

The Plumbers' Workbench interactive environment is implemented by a Presentation Manager interface on the client side which cooperates with the Pipeline Editor and the pipeline runtime support on the server side. The runtime support produces an interactive environment which shows how data flows in a multistream pipeline network and gives the pipeline programmer control over its progress.

All of the features described above are not provided in the initial shipment of the Plumbers' Workbench. The roll-out plan calls for two versions:

- Version 1 features the VMPIPE.EXE command as described above, and low-level functions it requires to access remote data and commands.
- The proposed enhancements for Version 2 include the pipeline editor and the interactive runtime support.

The Plumbers' Workbench is still in its infancy. It is likely to evolve as its users and we gain further experience.

1.1 Clients and Servers

The Plumbers' Workbench is a client/server application. The client runs on the user's workstation under the OS/2 operating system; the server runs in the user's virtual machine under the VM/CMS operating system.

A workstation can run multiple concurrent clients, which can be connected to a single server or to multiple servers. Only one server can run at any time in a virtual machine, but it can serve any number of clients concurrently, as shown in Figure 2.

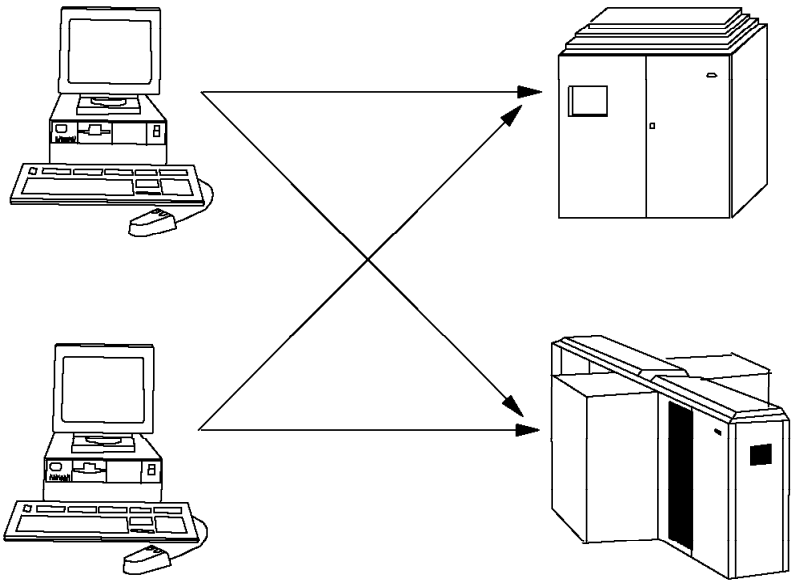


Figure 2. Multiple Clients May Access a Single Host

The Plumbers' Workbench is a client/server application, but it is one with a twist: Each user normally has a private server, which runs in the user's own virtual machine. No special privileges are required to run the server; any general user virtual machine can run it.

A user can employ a number of concurrent clients, which can be connected to a single server virtual machine or to multiple virtual machines. A user can start more than one server, but each server requires a separate virtual machine.

An installation can set up a common server (since any virtual machine can run the server code), but the security implications of an unattended server should be well understood before an installation sets up what could easily become a Trojan horse.

1.2 Implementation

The Plumbers' Workbench is written mainly in REXX. On the host, it relies on *CMS Pipelines* services; on the workstation it relies on a library of external functions, written in the C language. REXX is used in preference to the C language, because it is easier to write and debug and because REXX programs are easy to read. Programmer productivity was an important issue when the Plumbers' Workbench was written, because it was done on a small budget of time and money.

1.3 Why Did We Do It?

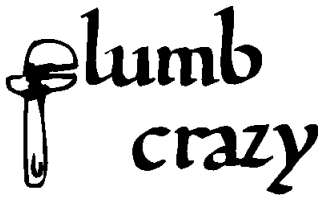
There are many ways to work with OS/2 and CMS, for example:

- You can move files using several means of file transfer.
- You can use the Network File System to access remote files.
- You can use REXEC to issue commands at a remote host.
- You can use EHLLAPI to pass commands through your LU2 session with the host.
- You will soon be able to use the Distributed Computing Environment (DCE) to create secure distributed applications.
- You will soon be able to use the CMS Graphical User Interface (the CMS GUI) to manipulate CMS objects using OS/2 metaphors.

So, why do you need yet another environment?

You need the Plumbers' Workbench because it integrates your use of OS/2 and CMS. Your OS/2 programs can work directly with CMS files and the responses to CMS commands. That is, the Plumbers' Workbench offers you the union of the facilities of OS/2 and CMS, not the lowest common denominator.

Part 1. User's Guide



The first part of this document contains the information you need to:

- Plan the installation of the Plumbers' Workbench
- Install the Plumbers' Workbench
- Customize the Plumbers' Workbench client and server
- Start and run the server
- Use the VMPIPE command to access host pipelines, issue host commands, and transfer files between the workstation and the host
- Understand the security implications of using the Plumbers' Workbench

Chapter 2. Installation Planning

This chapter discusses some decisions you must make before you can begin to install and configure the Plumbers' Workbench.

2.1 System Requirements

The Plumbers' Workbench will not bust your budget. For starters, it is free. Its client requires modest amounts of resources and its server runs in a standard virtual machine.

2.1.1 Hardware

Except for the network connection, no special hardware is required to support the Plumbers' Workbench. It will run with a normal 32-bit OS/2 workstation and with a normal CMS virtual machine that can run *CMS Pipelines*.

2.1.2 Software

In its initial version, the Plumbers' Workbench requires TCP/IP connectivity between the client and the server. No other special software is required.

2.1.2.1 Client

- OS/2 Version 3 (Warp) was used to develop and test the Plumbers' Workbench. The REXX feature must be selected when OS/2 is installed.
- TCP/IP for OS/2 Version 2.0 is required if the client is going to use TCP/IP to connect to the server. (Only the base kit is required.)

2.1.2.2 Server

- The Plumbers' Workbench uses the CMS callable services support for the data block interface, introduced in VM/Enterprise Systems Architecture Version 1 Release 2 Modification 0, to set the time stamp of files that are uploaded. The file time stamps are not preserved across an upload on earlier releases of CMS.
- *CMS Pipelines* must be at least at level 1.1.9 sublevel 33 (hex) to pick up the latest fixes for its TCP/IP support. This level is shipped as part of VM/Enterprise Systems Architecture Version 2 Release 1 Modification 0. It is also available to customers under the *CMS Pipelines* field test program.

2.2 Code Page Issues

The Plumbers' Workbench runs distributed over an ASCII system and an EBCDIC system; commands and text files might be entered in ASCII on the workstation for processing on the EBCDIC host. Thus, characters must be translated from ASCII to EBCDIC. (And files must be blocked and deblocked to insert or remove the carriage return/line feed sequences used on the workstation.)

Because CMS commands and pipelines are written in code page 1047 regardless of the language of your data files, the Plumbers' Workbench translates commands, pipelines, and file names between code pages 437 and 1047, irrespective of your code page settings for data files.

For text data being transferred between the workstation and the host, the Plumbers' Workbench determines the code page in use on the workstation through an application programming interface (the OS/2 function WinQueryCp). You can switch between two pre-specified code pages with the CHCP command. In contrast, CMS does not support a setable code page; you may specify the host code page to the Plumbers' Workbench through an environment variable, which is stored in the CMS global variable repository.

The Plumbers' Workbench performs all blocking and code page translation on CMS, because *CMS Pipelines* has powerful facilities for such conversion. Translation is supported between the code pages shown in Figure 3.

Figure 3. Country Extended Codepages

037	EBCDIC	United States, Canada, Netherlands, Portugal, Brazil, Australia, New Zealand.
273	EBCDIC	Austria, Germany.
274	EBCDIC	Belgium.
275	EBCDIC	Brazil.
277	EBCDIC	Denmark, Norway.
278	EBCDIC	Finland, Sweden.
279	EBCDIC	
280	EBCDIC	Italy.
281	EBCDIC	Japan.
282	EBCDIC	Portugal.
283	EBCDIC	
284	EBCDIC	Spain, Latin America.
285	EBCDIC	United Kingdom.
297	EBCDIC	France.
437	ASCII	PC Display: United States, Switzerland, Austria, Germany, France, Italy, United Kingdom.
500	EBCDIC	Belgium, Canada, Switzerland, International Latin-1. International number 5. This is the base codepage for *xlate*.
819	ASCII	ISO 8859 Latin Character Set 1 (Western Europe).
850	ASCII	PC Data-190: Latin Alphabet Number 1; Latin-1 countries.
863	ASCII	PC Display: Canada.
865	ASCII	PC Display: Denmark, Norway.
871	EBCDIC	Iceland.
1047	EBCDIC	C/370 variant of codepage 37, which takes into account the encoding of, for example, brackets that were inherited from the 3270 display system. This code page is used by Open Edition.

If you are using code page 437 on the workstation and code page 37 on CMS, you will not have much agony over code pages, though you might wish to use code page 1047 on CMS rather than 37, because this will make uploading of C language programs easier (the square brackets are translated incorrectly when you use code page 37 on the host).

However, if you are using one of the European code pages, it will be more difficult to select the correct translation to use. The question is, what will you be

uploading? Even if your terminal emulator is set to code page 277, for example, you might be uploading REXX programs, which will not be handled correctly if you select code page 277 on the host. (The reason is that REXX programs must be written in code page 37 or 1047.) Likewise, C language programs are closer to code page 1047 than to any other code page.

Where Specified: The workstation code page is specified by the CODEPAGE= statement in CONFIG.SYS or by the CHCP command. The CMS code page is specified by the CODEPAGE global variable in the PWB group.

2.3 Decide on Communications Protocol

For each pair of clients and servers, you must decide on the communications protocol to be used between them. You can choose between:

TCP/IP — Transmission Control Protocol/Internet Protocol. The Plumbers' Workbench uses connection-oriented (TCP) sockets.

Currently, your only choice is TCP/IP; we hope to support APPC in a future version.

2.3.1 TCP/IP

Just like other TCP/IP servers, the Plumbers' Workbench server must listen on a unique, "well-known" port to which its clients connect. The difference from a normal TCP/IP server is that each user will be running a dedicated server and thus, each user needs a dedicated port.

Though these ports can be managed on the "honor system", a security-conscious installation will consider pre-assigning port numbers, as described in 7.2.1.1, "Restricting TCP/IP Port Usage" on page 31.

Where Specified: The port is specified in the PWB_SERVER environment variable on the client and in the file PWB SERVER on the host.

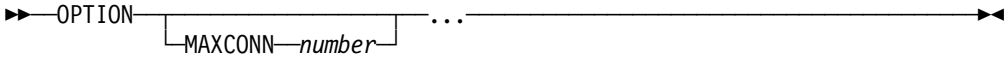
2.4 Running the Plumbers' Workbench Server Disconnected

You may decide to set up a central server for the Plumbers' Workbench. Before you do so, refer to Chapter 7, "Security" on page 29 for a discussion of the implications. We recommend that such a server not be given write access to critical data.

2.5 Setting Maximum Number of IUCV Connections

The server needs an IUCV connection for each client that is connected to it. Be sure to specify a large enough value for the maximum number of IUCV connections to the virtual machines that will be servers for the Plumbers' Workbench.

Where Specified: The number is specified in the OPTION statement of the user's entry in the CP directory.



The default is 10. Other IUCV or APPC activity in the virtual machine will require additional connections. 64 may be a “good” value.

Chapter 3. Installation

The Plumbers' Workbench is easily installed because it consists of user commands. Except for issues related to network access, installation of the Plumbers' Workbench does not require system-level changes to the host system; nor does it require command privileges beyond those for the general user.

The Plumbers' Workbench is available on diskette and by anonymous FTP from the International Technical Support Organization in Poughkeepsie. It is installed from diskette by copying the files in one directory to the hard disk and uploading the files in another directory to the host. Installation from the FTP server is accomplished by uploading the server part or downloading the client part, depending on where the files were transferred.

3.1 Installation from Diskette

Note: Before beginning to install the Plumbers' Workbench, read the contents of the README.1ST file in the root directory of the diskette.

3.1.1 Directory Structure

The Plumbers' Workbench diskette distribution contains one directory (SG244523) in the root directory. The diskette contains these directories:

```
a:\sg244523
        \client
        \server
        \samples
        \src
```

The `client` directory contains the files that are required to run the client part; the `server` directory contains files that should be uploaded to the host; the `samples` directory contains sample REXX filters for processing OS/2 data; and the `src` directory contains source files and other useful information.

3.1.2 Installation Tasks

Installation is performed by moving the OS/2 files from the diskette to the hard disk and by uploading the server code and samples.

You can move the files using several methods; these steps are one way to install the Plumbers' Workbench from a diskette:

1. Insert the diskette in the diskette drive.
2. Log on to your virtual machine through the Communications Manager or PC3270. The examples below assume you are using session A.
3. On CMS, create a directory to store the files that make up the server and access this directory:

   ```
   create directory .pwb
   access .pwb f
   ```

4. From OS/2, issue these commands to upload the server to the newly created directory. Be sure to do a binary upload (that is, without translating from ASCII to EBCDIC):

```
almcopy a:\sg244523\server\*.rex ha: = rexx f /bin
almcopy a:\sg244523\server\*.exc ha: = exec f /bin
```

(ALMCOPY is a utility that is supplied with the PC3270; you may have a different favorite method for file transfer.)

5. From OS/2, issue these commands to create the target directory and move the executables to the target directory:

   ```
   mkdir \pwb
   xcopy a:\sg244523\client\*.* c:\pwb
   ```

   The target directory must exist in the PATH before you can use the Plumbers' Workbench client. You can add the newly created directory to the PATH system environment variable, which is set in CONFIG.SYS, or by the PATH command. Alternatively, you can move the files to a directory that is already in the PATH.

   Take Note!

   The release of Plumbers' Workbench delivered with this publication requires the PWBCTL.CMD file to be located in the c:\jph\src\pwb\ directory. This is a strict requirement.

Chapter 4. Customization

Customization tasks make the server and the client ready for each other. You must tell the client where to find the server and you must tell the server which clients it should obey. The dictum not to speak to strangers applies just as much to the Plumbers' Workbench.

4.1 Customizing the Client's Environment Variables

The client reads several environment variables (Figure 4) to determine where to connect, where to write error messages, and where to write trace data. You can set the variables permanently in your CONFIG.SYS file or you can set them temporarily in a command file that invokes the Plumbers' Workbench. You can even set them "by hand" before invoking a function of the Plumbers' Workbench.

Figure 4. Overview of Client Environment Variables

Variable	Default	Usage
PWB_MESSAGEFILE	The standard error output.	Specify the file to receive messages that are issued by the Plumbers' Workbench client.
PWB_SERVER	None. This variable must be set.	Specify the location of the Plumbers' Workbench server that you wish to connect to.
PWB_TRACEFILE	The standard error output.	Specify the file to receive trace information.
PWB_TRACELEVEL	1	Specify the level of detail of the reporting of activity within the Plumbers' Workbench client.

4.1.1 PWB_MESSAGEFILE

The environment variable PWB_MESSAGEFILE specifies the path to use when writing messages to the user. If the variable is not set, the messages are written to the standard error file.

4.1.2 PWB_SERVER

The environment variable PWB_SERVER must be set up to contain a character string of blank delimited words that specify the authorizations, the protocol, and the location of the server.

TCP/IP:

1. Authorization level for the partner, as defined in 7.1.1, "Authorizations" on page 29.
2. The protocol to use for access to the server. Specify "tcp".
3. The local port to use. Specify a particular port if the server has been or will be set up to accept connections from a particular port only. This word is usually specified as the number 0, which means that TCP/IP assigns the port number for the outgoing connection request.

4. The host address of the system where the server is running. You can specify this as a domain name, such as pucc.princeton.edu, or as a dotted-decimal number, such as 9.12.14.168.
5. The server port number. Specify the number you have been assigned.

These options are shown in Figure 5.

```
set pwb_server=a tcp 0 wtscpok.itsc.pok.ibm.com 1995
set pwb_server=a tcp 0 9.130.8.1 1995
```

Figure 5. Two Ways to Designate the Server

4.1.3 PWB_TRACEFILE

The environment variable PWB_TRACEFILE specifies the path to use when writing tracing information. If the variable is not set, the messages are written to the standard error file. We recommend the use of a file, particularly when module tracing is enabled.

4.1.4 PWB_TRACELEVEL

The environment variable PWB_TRACELEVEL specifies the amount of detail reported by the Plumbers' Workbench client while it is running. The trace level should be specified as a non-negative number. Trace level 0 suppresses trace information. The information written for higher levels is unspecified. The default trace level is 1, which writes summary trace information. Larger values will produce progressively more trace information; use a larger number only when you have been asked to do so to diagnose a problem with the Plumbers' Workbench.

4.2 Customizing the Server

For the server, you must create two files. The first file contains information about the communications endpoints on which the Plumbers' Workbench server should listen for connection requests. The second file contains information about the clients that are authorized to use your server. The contents of these files are explained below.

Both files contain readable EBCDIC; you can use XEDIT to maintain them. Lines that begin with an asterisk are ignored (that is, they are comments), as are entirely blank lines.

4.2.1 Specifying Protocols and Endpoints

The type of communications protocol to use and the specific endpoints to monitor are specified in the file PWB SERVER. Each non-comment line of the file specifies a communications endpoint to be monitored. There is no reason to set up more than one endpoint for a particular protocol type.

The first word of each line identifies the protocol:

tcp TCP/IP.

The second word specifies the port number on which the Plumbers' Workbench server will listen for connection requests.

An example of a completed file showing port 1995 for protocol tcp is shown in Figure 6 on page 15.

```
tcp 1995
```

Figure 6. A Sample PWB SERVER File

4.2.2 Specifying Authorized Clients

Authorized clients are specified in the file PWB CLIENTS. Each non-comment line of the file specifies a client or a group of clients. The first two words specify the authority to be extended to the client and the type of protocol. The contents of the remainder of the line depend on the particular protocol.

TCP/IP: Each line contains these words:

1. The authorization level to extend to the client. Refer to 7.1.1, “Authorizations” on page 29.
2. The keyword “tcp”.
3. The internet address of the client or a mask of valid addresses. This is specified as a single asterisk, or as dotted-decimal notation, or in dotted-decimal with asterisk wildcards.
4. A port number or an asterisk.

Authorizations are tested in the order they are written in the file. To assign a different authorization to a subset (for example, your own workstation), put the more restrictive address first. A sample authorization file is shown in Figure 7.

```
* First word is authorization level
* Second port is the protocol name; the rest of the line depends on this
* For tcp,
*          the third word specifies the dotted-decimal host address of
*                         the client.
*          the fourth word specifies the port number.
* An asterisk (*) is wildcard in these.
* My own workstation can do everything:
a tcp 9.12.14.168  *
* Scott can write:
w tcp 9.12.14.5 *
* The rest of the ITSO can read:
r tcp 9.12.14.*    *
* Everybody can send a message:
n tcp *            *
```

Figure 7. A Sample PWB CLIENTS File

4.2.3 Customizing the Server's Environment Variables

Host environment variables are read from the CMS global variable repository, which is maintained by the GLOBALV command. The Plumbers' Workbench variables must be stored in the group PWB. In the following description, the variable name is prefixed with PWB. to remind you that the variables are in a separate group.

4.2.3.1 PWB.CODEPAGE

The environment variable PWB.CODEPAGE specifies the code page of the files on the host. If you are transferring programs, you might prefer to use code page 1047 rather than the code page for your country. The GLOBALV command is shown in Figure 8.

```
globalv select pwb setp codepage 1047
```

Figure 8. Setting the Host Code Page

4.3 Installation Verification Procedure

Now try a simple request to see if you have got everything set up correctly. First start the server by issuing the PWB command on your CMS session. Then issue a request from the workstation. Figure 9 shows a successful installation verification.

Figure 9. Installation Verification

Client	Server
	pwb
[C:\JPH\SRC\PWB]vmpipe /cmd query cmslevel **PWB0048I: Plumbers' Workbench level 1.0000 dated 19950715.**	
	Request from port 1133 on 9.12.14.168 authorised a **PWBPWB0048I Plumbers' Workbench level 1.0000 dated 19950715.** **200912 PWB 1.0000 19950715 CMS a 1047** **200912 Partner is: PWB 1.0000 19950715 OS/2 a 437**
CMS Level 11, Service Level 502	
	PWBPWB0047I Partner terminates: No more work to do. Bye....

Chapter 5. Running the Plumbers' Workbench Server

The Plumbers' Workbench server is started by running an EXEC. This EXEC monitors the communications endpoints for which it has been customized. It also enables two immediate CMS commands.

5.1 Starting the Server

The server is started by the PWB EXEC. PWB runs until it is terminated by the immediate command STOP.

You may wish to issue the commands in Figure 10 before starting the server.

```
cp spool console start to *
cp terminal more 0 0
cp set msg off
```

Figure 10. Suggested CP Commands to Use with the Server

The first command keeps a log of the messages that are written to your terminal. The second command prevents the terminal from hanging the server when the screen is full of these messages. The third command prevents the screen from going into MORE... when someone sends the server a message (because the message will be rejected).

5.2 Immediate Commands

Two immediate commands are intended for general use.

5.2.1 CMS—Issue a CMS Command

Specify the command to issue after the immediate command verb. Note that the server waits while the command runs. It will resume when the command completes. A ready message is displayed when the command completes.

Note: The command is issued through the *CMS Pipelines* host command interface, *subcom cms*. Thus, the pipeline that controls the server is not dispatched while the command is running.

5.2.2 STOP—Terminate the Plumbers' Workbench

Issue the STOP command to terminate the server gracefully. All current requests are processed, but no new connections are accepted.

Use PIPMOD STOP to terminate a hanging TCP/IP session.

5.3 File Upload and Download Commands

The Plumbers' Workbench recognizes two other immediate commands, which it also forwards to CMS. These commands are issued as part of file upload and download. Passing these commands to CMS allows the Plumbers' Workbench to co-exist with old-fashioned file upload and download.

ALMCOPY issues a command that has EXEC as the first word. The EXEC immediate command deletes the word EXEC before it passes the actual command to CMS's full command resolution.

SEND and RECEIVE commands and the Communications Manager use IND$FILE. The IND$FILE immediate command puts the command verb back in front of the command it issues to CMS.

Chapter 6. VMPIPE—The Command Interface for the Plumbers' Workbench

With the VMPIPE command, you can write REXX programs that:

- Connect pipelines on the host to the client's standard input or standard output, or to both
- Issue CMS commands on your host session and pass the response to standard output
- Upload and download files

A conceptual overview of VMPIPE command interaction with host plumbing services is shown in Figure 11.

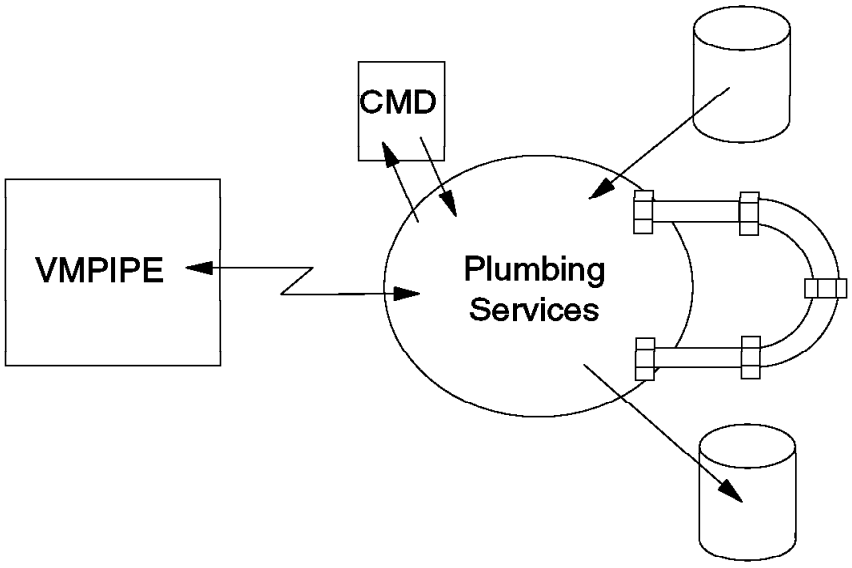


Figure 11. VMPIPE Allows Extensive Host Interaction

All requests are issued through the VMPIPE command, using one of two forms. The first form is suitable for issuing a pipeline on CMS; the argument string specifies the pipeline to run on the server. This is a front end to the general VMPIPE command, which is described in section 6.2.

The VMPIPE command is a normal OS/2 command, and as such, it (or rather OS/2) supports piping and redirection. Thus, be careful to enclose the pipeline to be run on the host within double quotes (″). If you forget, OS/2 will attempt to run the pipeline itself, most likely unsuccessfully.

6.1 Vanilla VMPIPE—Run a Pipeline on Host

When the VMPIPE command is issued without specifying any of the “mocha” options, it passes its standard input data through a pipeline on the server and then passes the output from the pipeline to its standard output, as shown in Figure 12.

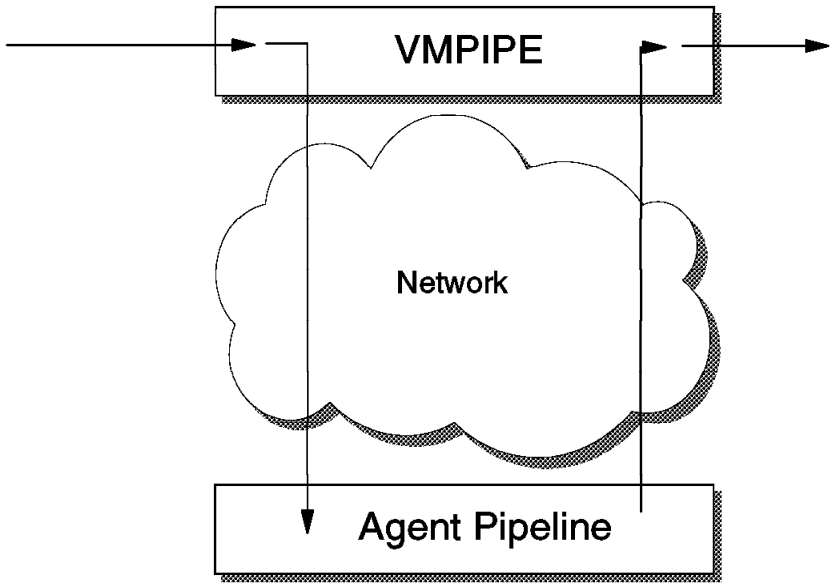
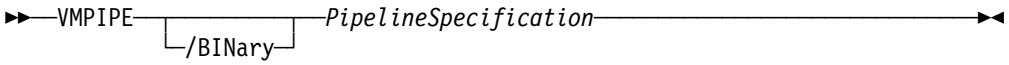


Figure 12. A Vanilla VMPIPE

The command syntax is as follows:



Argument String: The argument string following the /BINARY option specifies the pipeline segment to be run on the server. You can use one of three styles when specifying this pipeline:

- A simple sequence of stages that does not begin with a left parenthesis and whose first word is not CALLPIPE. Such a “straight” pipeline segment is specified without connectors; a connector will be added to each end of the pipeline automatically.
- Global options in parentheses followed by a pipeline specification. The global options can specify an end-character, which allows for a multistream pipeline specification. Connectors will be prefixed and suffixed to this pipeline specification. Use *fanout* to tunnel the data from an internal point in the multistream pipeline to the end of the pipeline specification. Do not specify the stage separator option; the default stage separator is required for the pipeline to work.
- A complete subroutine pipeline, which begins with the word CALLPIPE. You can specify all global options, including an end-character and even a stage separator. You must specify the connectors as well. The input from the

client will be passed on the primary input stream; the output written to the primary output stream will be returned to the client. This style is the most flexible, but it incurs the overhead of running a REXX filter to set it up.

Operation: The pipeline is translated from ASCII code page 437 to EBCDIC code page 1047 before it is passed to the host pipeline.

Unless the /BINARY option is specified, the contents of the standard input are translated from ASCII to EBCDIC when they arrive at the host and the contents of the output from the host pipeline are translated back to ASCII when they leave the host. The uploaded byte stream is split into records at line ends when it arrives at the host. The records sent back from the host will have the line end sequence (carriage return and line feed) appended to them.

When /BINARY is specified, the contents of the standard input are transmitted as a byte stream without conversion. The length of the records flowing into the host pipeline is unspecified; your pipeline must deblock the data stream according to its format. Record boundaries are ignored when the result is transmitted back to the workstation.

Notes:

1. The pipeline must be enclosed in double quotes (″) to prevent it from being scanned by the OS/2 command shell.
2. Messages that are issued by *CMS Pipelines* are written to the terminal or console SPOOL of the server, or to both. But they are not returned to the client.
3. The pipeline is issued using the ADDPIPE pipeline command on the server. A non-zero return code from the ADDPIPE pipeline command is reported back to the client as an error, but errors that occur after the pipeline specification has been added to the server's pipeline set are not reported to the client.
4. CMD.EXE has a limit of 299 characters of input. More complex pipelines can be stored as subroutine pipelines, which are uploaded (or created on the host) or specified using the /FILE option to be described below.
5. The *CMS Pipelines* stages in the simple VMPIPE parameter list cannot be stages that must be first in a pipeline, because the server prefixes the pipeline with the stream that contains the data that is piped into VMPIPE. You can use *preface* and *append* to run a device driver that must be first in a pipeline.

Examples: Figure 13 shows how to read an OS/2 file and process it through a host pipeline:

```
[C:\JPH\SRC\PWB]vmpipe <\config.sys ″xlate lower , blank|count words″
PWB0048I: Plumbers' Workbench level 1.0000 dated 19950715.
266
```

Figure 13. Plumbing OS/2 File for Host Processing

Figure 14 on page 22 shows how to use *append* to run a device driver that must be first in a pipeline. It also shows how to access a host database. We had run SQLINIT in the server virtual machine prior to issuing this command.

```
[C:\JPH\SRC\PWB]vmpipe "append sqlselect * from sqldba.projects"
PWB0048I: Plumbers' Workbench level 1.0000 dated 19950715.
PROJECT_NAME--- START_DATE END_DATE-- AUTHORIZED----------------
BLUE MACHINE    1988-08-15 1988-12-16 1988-06-30-16.00.29.000000
GREEN MACHINE   1990-04-27 1990-12-18 1990-01-26-10.00.03.000299
ORANGE MACHINE  1989-07-10 1990-06-04 1988-10-11-09.45.00.000000
RED MACHINE     1988-02-01 1990-01-12 1987-12-14-08.30.01.000000
WHITE MACHINE   1988-09-19 1988-11-11 1987-11-03-18.34.02.000999
```

Figure 14. Using APPEND When Device Driver Must Be First

Figure 15 shows how to inject a line into an OS/2 pipeline and how to process this line as a CP command on the host.

```
[C:\JPH\SRC\PWB]echo q time|vmpipe "cp | xlate lower"
PWB0048I: Plumbers' Workbench level 1.0000 dated 19950715.
time is 20:21:25 edt tuesday 08/01/95
connect= 70:18:43 virtcpu= 014:01.03 totcpu= 014:35.14
```

Figure 15. OS/2 Command Output As Host Pipeline Input

Note that, in the example above, the first stage separator applies to the OS/2 pipeline, but that the second one is passed to the VMPIPE command. This happens because the second stage separator is within quotes.

Be sure to remember the quotes. Figure 16 shows what happens when the VMPIPE arguments are not enclosed in quotes (or rather, what happens when OS/2 and you disagree about what constitutes the argument string for VMPIPE).

```
[C:\JPH\SRC\PWB]echo q time|vmpipe cp | xlate lower
SYS1041: The name specified is not recognized as an
internal or external command, operable program or batch file.
SYS1092: The handle could not be duplicated during a pipe operation.

The external process was cancelled by a Ctrl+Break or another process.
```

Figure 16. Mistakes Can Happen When You Forget Quotes

OS/2 was unable to resolve the XLATE command, which it tried to run as a third process for the pipeline.

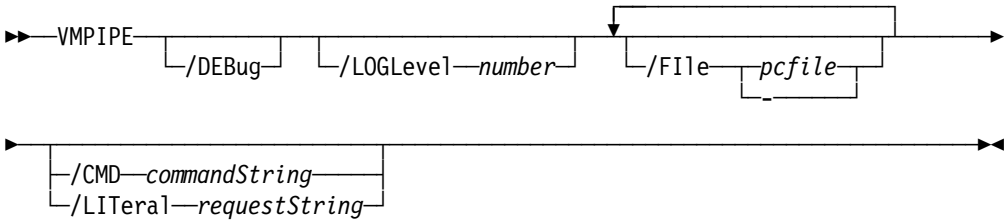
Figure 17 on page 23 shows how to specify a multistream pipeline to be run on the host. The pipeline splits all assignments in the CONFIG.SYS file at semicolons and prefixes the variable's name to each of the parts. Only the first ten lines of the response are returned to the workstation. Multistream pipelines may not be the easiest reading, even when they are formatted to show their topology. This style is clearly not to be recommended.

```
[C:\JPH\SRC\PWB]vmpipe <\config.sys "callpipe (end \) *:|xlate lower|find set_|s
pec 5-*|c:chop after =|j:juxtapose|take 10|*:\c:|split ;|j:"
PWB0048I: Plumbers' Workbench level 1.0000 dated 19950715.
user_ini=c:\os2\os2.ini
system_ini=c:\os2\os2sys.ini
os2_shell=c:\os2\cmd.exe
autostart=programs,tasklist,folders,launchpad
runworkplace=c:\os2\pmshell.exe
comspec=c:\os2\cmd.exe
path=e:\toolkit\som\bin
path=e:\ibmwf\dll
path=e:\ibmcpp\bin
path=e:\toolkit\bin
```

Figure 17. A Multistream Pipe Run on the Host

6.2 Mocha VMPIPE—Advanced Options

The real VMPIPE command supports several request types and the batching of requests into files.



Specify command options by a leading slash (/). The slash must be prefixed to the option; no blanks are allowed after the slash. Case is ignored in command options, but it is retained in the value that follows an option.

Of the command options, /DEBUG must be specified first, because it is acted upon by the C language program that is invoked for the VMPIPE command.

/DEBUG Enable extensive trace in the C language code that supports the Plumbers' Workbench. The trace output is written as specified in the PWB_TRACEFILE environment variable.

/LOGLEVEL Specify the level of detail to be written to the log file. This option is a convenience for setting the PWB_LOGLEVEL environment variable.

/FILE Read requests from a file before continuing the processing of further options. The next word specifies the file path. A hyphen (-) indicates that the requests should be read from the standard input. Each input line is processed as a request, as defined in 6.2.1, "Request Types" on page 24.

/CMD Issue a CMS command using full CMS command resolution and pass the response back. The remaining argument string is passed to the server without looking for further options. Note that CP responses are not returned. (That is, the /CMD option is equivalent to /LITERAL CMD.)

/LITERAL The remainder of the argument string is taken as a single request, as defined in 6.2.1, "Request Types" on page 24.

If neither /CMD nor /LITERAL is specified in the argument string before the first word that does not begin with a slash, the remainder of the argument string is issued as a pipe request in the same way that the simple variant of the VMPIPE command does. The /BINARY option is also recognized.

Operation: The argument string is processed, building requests from the arguments and the contents of files. The requests are issued when the argument has been scanned completely. The requests are processed in parallel, and therefore will not always complete in the order they arrive. You should be aware of this asynchronous behavior as you develop your application.

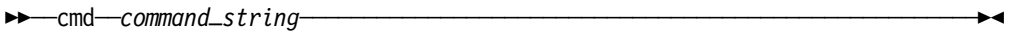
The requests are translated from code page 437 to code page 1047 when they arrive at the host, irrespective of the code page settings. The data streams are translated according to the format specified, the workstation code page, and the host code page setting.

6.2.1 Request Types

You can issue the requests described below. All requests should be issued using ASCII encoding for the request type and the operands. Refer to 6.2.2, “Data Conversion” on page 25 for information about how to specify conversion when data is sent between the partners. Refer to 6.2.3, “File Naming Conventions” on page 26 for information about how to specify a host file name.

6.2.1.1 cmd

Issue a CMS command and pass the response back on the standard output. The remainder of the line contains the command. The output from the command is translated to ASCII and a carriage return and a line feed character are appended to each line.

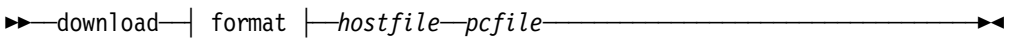


Example:

```
cmd query search
```

6.2.1.2 download

Download a file. A file is copied from CMS to the workstation.



The first word specifies the format to be used for transmission. Note that this operand is required. Refer to 6.2.2, “Data Conversion” on page 25.

The second word specifies the host file in the standard file name format. Refer to 6.2.3, “File Naming Conventions” on page 26.

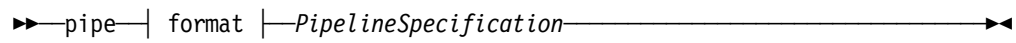
The third word specifies the path to the file as it is to be stored on the workstation. Any existing file is erased.

Example:

```
download binary nxpipe.module e:\tmp\nspipe.vmm
download pctext 4523msgs.script 4523msgs.scr
download pctext piper/pwb/pwbctl.rexx \tmp\pwbctl.rex
download pctext a/profile.exec /tmp/profile.exc
```

6.2.1.3 pipe

Pass data through a host pipeline. The contents of the standard input are passed through the pipeline on the server; the output from the pipeline is then passed to the standard output.



The first word specifies the format used for transmission, both to the server and from the server. Note that this operand is required. Refer to 6.2.2, “Data Conversion”.

The remainder of the command is passed to the server as the pipeline to be issued.

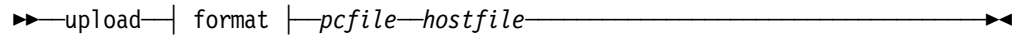
Example:

```
pipe binary xlate from 437 to 1047
pipe pctext book2os2
```

Note that there is no redirection at the level of the individual requests; thus, only the first request can read from standard input; the second and subsequent requests will see end-of-file on the standard input.

6.2.1.4 upload

Copy a file from the workstation to the server.



The first word specifies the format to be used for transmission. Note that this operand is required. Refer to 6.2.2, “Data Conversion”.

The second word specifies the path to the file to be uploaded.

The third word specifies the host file in the standard file name format. Refer to 6.2.3, “File Naming Conventions” on page 26. Any existing file is erased.

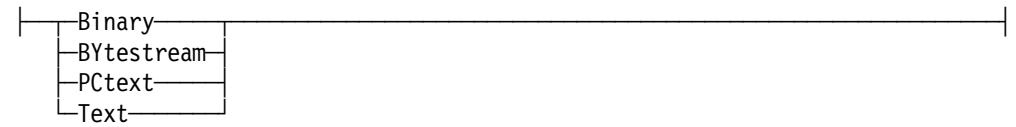
Example:

```
upload binary e:\tmp\nspipe.vmm nxpipe.module
upload pctext 4523msgs.scr 4523msgs.script
upload pctext \tmp\pwbctl.rex .piper/pwb/pwbctl.rexx
```

6.2.2 Data Conversion

You must specify the format of the data that is sent between the partners.

format:



BINARY No conversion is required.

BYTESTREAM A synonym for BINARY.

PCTEXT Convert between ASCII and EBCDIC (or the reverse, as appropriate). Use carriage return and line feed to terminate lines. When receiving data from the workstation, terminate before the first end-of-file character (control-z).

TEXT Convert between ASCII and EBCDIC (or the reverse, as appropriate). Use line feed to terminate lines. This style is the one used for text files in UNIX systems.

Note that there is no leading slash in the keywords.

6.2.3 File Naming Conventions

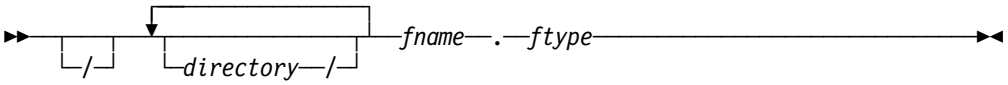
File names, be they host or workstation, are named using a portable file naming scheme, which is defined in the POSIX standard. Directories in the path to a file are separated by slashes (/) (rather than backslashes, as used by PC-DOS).

This makes little difference for OS/2 files, except that a slash will be changed to a backslash. The drive letter is considered part of the top-level directory.

The CMS file naming scheme is mapped into a one-word file name by:

- Juxtaposing the file name and the file type with a period between them as in PROFILE.EXEC
- Changing the periods in an SFS directory path to slashes.
- Prefixing the SFS directory path and a slash to the file name and file type as in SERVER8:PIPER/PROFILE.EXEC.
- Treating a file mode as a one-letter directory name as in A/PROFILE.EXEC.

Ensure that the file name is of this form:



6.2.4 Examples

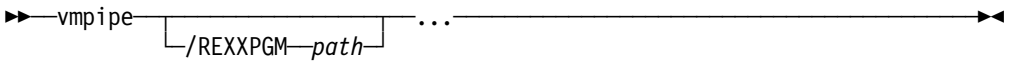
To query the status of file mode A, use the VMPIPE command shown in Figure 18.

```
[C:\JPH\SRC\PWB]vmpipe /cmd q disk a
PWB0048I: Plumbers' Workbench level 1.0000 dated 19950715.
LABEL  VDEV M  STAT   CYL TYPE BLKSZ   FILES  BLKS USED-(%) BLKS LEFT  BLK TOTAL
SCRTCH 192  A   R/W    30 3390 4096       18         63-01       5337       5400
```

Figure 18. Run a CMS Command Using VMPIPE

6.3 Service Aids

The C language layer of the VMPIPE command also supports an option to specify a REXX program to be run.



This option is useful for testing of the external functions that the VMPIPE command declares.

```
[C:\JPH\SRC\PWB]vmpipe /rexxpgm rexxtry say PwbCodepage()
437
  ..... rexxtry.CMD on OS/2
```

Chapter 7. Security

Security should be one of your concerns when you are setting up the Plumbers' Workbench.

Hackers on the Internet have received much media attention, which has led to TCP/IP networks getting an unjustified reputation for lack of security. While it is true that TCP/IP does not attempt to address many security issues, it is not inherently less secure than an SNA network. After all, your SNA network would be equally open to attack if it were run in an environment where the lines and network controllers were not under control of your organization.

7.1 Understanding the Networking Implications of the Plumbers' Workbench

The Plumbers' Workbench client and server communicate through a pair of TCP/IP connection-oriented sockets. The connection is initiated from the workstation (the client).

For the TCP/IP protocol, the server accepts or rejects the connection request based on the origin of the conversation.

Data flows in clear text between the partners. Each partner performs services on behalf of the other.

That is, there are no smarts in the network layer of the Plumbers' Workbench. Its data path is established without performing a logon to the system where the server runs.

7.1.1 Authorizations

A rudimentary level of authorization checking is done by the Plumbers' Workbench when it processes a request from its partner. This checking is based on a single authorization level, which is established when the connection is set up. For the client, the authorization is a parameter that is specified along with the network address (or resource name) of the server. The server maintains a table of acceptable origins for session requests and their associated level of authorization.

The authorization levels defined are:

n No access right is extended to the partner. Messages will be accepted and processed as appropriate. File I/O, pipelines, and commands are rejected.

r File read authority is extended to the partner.

w File write authority is extended to the partner.

c Pipeline access rights and command rights are extended to the partner. The Plumbers' Workbench will issue pipeline specifications and system commands on behalf of the partner.

a All rights are extended. Currently, there are no rights defined beyond command rights; this designation is "for future use", but it is OK to use now.

Rights at a particular level imply the rights of levels above. For example, file write permissions imply file read permissions.

Clearly, the Plumbers' Workbench is designed to be run under control of the user. On the host, the server will normally run in the user's virtual machine and thus assume the user's identity as far as the host operating system is concerned.

If the server is run in a virtual machine that has been granted more privileges than a general user would normally have, all workstations that are allowed to connect to this server will be extended these privileges. That is, unless controlled carefully, the server can become a Trojan horse.

The distinction between write, command, and all authority may not be enforceable on all host operating systems. On VM/ESA, a malicious user who has command privileges can replace the REXX code that runs the server thread for future connection requests. Since the Plumbers' Workbench authorization checking is performed in this REXX code, it can clearly be subverted by a user who has command privileges. Likewise, a user who has write privileges may be able to replace the file listing authorized users and thus obtain higher privileges in future sessions.

A client that has read privileges cannot modify data at the server, but it can read the file that contains the list of acceptable session origins and thus discover the identity of systems that it might be profitable to penetrate.

7.2 Network-Specific Issues

The following sections describe issues that are specific to the protocol used to communicate between the client and the server.

7.2.1 TCP/IP

The Plumbers' Workbench assumes that you have a trustworthy network with trustworthy users. For example, it is assumed that a user does not change the IP address assigned to the workstation to masquerade as some other workstation and by implication as some other user.

You must deal with two issues when considering whether to use TCP/IP to connect the Plumbers' Workbench client to its server:

1. How can the server be sure that it is serving a workstation that is under control of the user, and
2. How can the user of a workstation be sure that the connection is indeed to their host account.

The server uses a list of authorized IP addresses to determine whether the connection request comes from an authorized workstation.

Since the client connects to a TCP port on the host system, it must be able to determine that this port is indeed being listened on from the user's virtual machine.

The user may be able to determine that the PWB command is running and listening on the correct port, but for use in a disconnected virtual machine, the port should be restricted by the system support staff.

7.2.1.1 Restricting TCP/IP Port Usage

The installation can assign pre-defined port numbers to users of the Plumbers' Workbench. These port numbers should be reserved in a PORT statement. The PORT statement can be placed in the PROFILE TCPIP file or in some other file that is subject to an OBEYFILE command, which will activate the restriction. A sample of the PORT statement is shown in Figure 19.

```
   PORT  1995 TCP  PIPER   NOAUTOLOG
```

Figure 19. TCP/IP PORT Control Statement

You may consider omitting the NOAUTOLOG operand, but do so only after careful analysis of the security and operational implications.

Allowing the automatic logon of the user's virtual machine may allow unauthorized access to go unnoticed, while it might be noticed by a user who is logged on and monitors the console of the virtual machine.

But users might not like this aspect of TCP/IP: When TCP/IP is restarted, it checks after some time whether the virtual machines that have ports assigned are listening to these ports. If they are not, TCP/IP will assume that the server is hung, force the virtual machine off and log it back on. A user who is in the middle of an XEDIT session is unlikely to appreciate such a favor.

We recommend that you specify NOAUTOLOG.

Chapter 8. Messages

The Plumbers' Workbench client issues the messages that are described below. A common message repository is used for the client and for the server. But remember that the server relies heavily on *CMS Pipelines*, which has its own messages. For example, problems with TCP/IP on the client side are reported using the messages described here, but on the server side the problem is likely to be reported by *CMS Pipelines*.

8.1 Messages Sorted by Number

1E Unsupported protocol ***sub-1*****.**

Explanation: The second argument word to PWBMAIN.EXE specifies the type of communications protocol to use to connect to the server. It is not recognized as a valid type. The valid types are:

TCP Transmission control protocol.

System Action: The client terminates.

2E Error running REXX program ***sub-1*****. Return code is** ***sub-2*****.**

Explanation: The REXX program that implements the client of the Plumbers' Workbench ended with the return code shown. The return code can be due to an error detected by the Plumbers' Workbench, or it can be due to an error discovered by the REXX interpreter. The reason may be that the REXX program is not installed correctly.

User Response: If no other messages have been issued and displayed in dialog windows, refer to the standard output of PWBMAIN. This is usually discarded by the operating system. Invoke the Plumbers' Workbench's client program from a command line to store the standard output and the standard error file:

```
[C:\PWB] pwbmain >e:\tmp\pwbmain.out 2>&1 a tcp 0 wtscpok.ibm.com 1995
```

System Action: The client terminates.

3E Too few arguments in function call. ***sub-1*** **received;** ***sub-2*** **expected.**

Explanation: The Plumbers' Workbench's REXX function package detected an incorrect call to a function. Not enough arguments are supplied on the function call.

User Response: Refer to the REXX error messages on the standard output.

System Action: The function call is rejected; REXX signals a syntax error.

4E Unexpected REXX interface. Result string is ***sub-1*** **bytes;** ***sub-2*** **are required.**

Explanation: The REXX interpreter does not adhere to its published Application Programming Interface. The result string provided for an external function is shorter than 33. It should be 256.

User Response: Look for corrective service.

System Action: The function call is rejected; REXX signals a syntax error.

5E Integer argument expected; received ***sub-1*****.**

Explanation: An integer is expected for an argument to an external function in the Plumbers' Workbench, but the argument was not processed completely by the `strtol()` function.

User Response: Look for corrective service.

System Action: The function call is rejected; REXX signals a syntax error.

6E Unrecognised command verb *sub-1***.**

Explanation: A service request at the base protocol level is issued for a service that is not recognized.

System Action: The request is ignored.

7I Message discarded for unused port *sub-1* **from** *sub-2* **sequence** *sub-3* **cmd** *sub-4***.**

Explanation: The base protocol level receives a transmission destined to an unassigned sub-port. This might indicate that the client and the server are out of synchronization. Or it might indicate that the partner is not reacting correctly to an error packet.

User Response: Look for corrective service.

System Action: The packet is discarded.

8E Too many arguments in function call. *sub-1* **received;** *sub-2* **expected.**

Explanation: The Plumbers' Workbench's REXX function package detected an incorrect call to a function. Too many arguments are supplied on the function call.

User Response: Refer to the REXX error messages on the standard output.

System Action: The function call is rejected; REXX signals a syntax error.

9E File *sub-1* **not found.**

Explanation: A file to be read does not exist.

System Action: The requested data transfer is not performed.

10E Unable to open file *sub-1***.**

Explanation: A file cannot be opened. On write, it may be that the target directory for the file does not exist.

System Action: The requested data transfer is not performed.

11E Syntax raised on line *sub-1* **of** *sub-2***.**

Explanation: A syntax error occurred in the REXX program. This is a programming error in the Plumbers' Workbench.

User Response: Refer to the REXX error messages on the standard output. Look for corrective service.

System Action: The function call is rejected; REXX signals a syntax error.

12E Novalue raised on line *sub-1* **of** *sub-2* **for variable** *sub-3***.**

Explanation: An unset variable was referenced in the REXX program. This is a programming error in the Plumbers' Workbench.

User Response: Refer to the REXX error messages on the standard output. Look for corrective service.

System Action: The function call is rejected; REXX signals a syntax error.

13E C Library error: *sub-1* **(in** *sub-2***).**

Explanation: The library has detected an error. The system error string is substituted.

System Action: Depends on the calling routine.

15E Integer expected. *sub-1* **was specified.**

Explanation: An integer argument is expected, but it is not converted by the `strtol()` library function.

System Action: Depends on the calling routine.

16E Integer is too small. *sub-1* **was specified; the minimum is** *sub-2***.**

Explanation: An argument is converted to integer without error, but the number specified is less than the smallest acceptable number.

System Action: Depends on the calling routine.

17E Integer is too large. *sub-1* **was specified; the maximum is** *sub-2***.**

Explanation: An argument is converted to integer without error, but the number specified is greater than the largest acceptable number.

System Action: Depends on the calling routine.

19E Unable to initialise socket interface.

Explanation: The `sockinit()` function returns a non-zero value.

User Response: Check your TCP/IP setup. Did you intend to use TCP/IP to connect to the server?

System Action: The client terminates.

20E Internet address *sub-1* **is not correct format.**

Explanation: The first character of the host address is a number, but the remainder of the string is not in the correct format for an internet address in dotted-decimal notation. (`inet_addr()` returns -1.)

System Action: The client terminates.

21E Lost connection to *sub-1* **port** *sub-2***.**

Explanation: An operation to read or write to the partner is rejected by the communications layer. (EPIPE is set by TCP/IP.)

System Action: The client terminates.

22E ERRNO *sub-1* **on socket call.**

Explanation: An error is returned from a socket function. The error number is substituted.

System Action: The decoded error information is written to the standard error file. The client terminates.

23E Partner terminated session prematurely.

Explanation: The partner closed the session without sending the end-of-session packet. It is likely that the partner has terminated because of a programming error.

System Action: The client terminates.

24E Too few parameters. *sub-1* **specified;** *sub-2* **are required.**

Explanation: The TCP/IP transport layer requires four parameters, but fewer are specified.

System Action: The client terminates.

25E Authorities must be a single letter. *sub-1* **characters found.**

Explanation: The first word of the arguments to PWBMAIN does not contain a single character.

System Action: The client terminates.

26E Authorities must be n, r, w, c, or a. *sub-1* **found.**

Explanation: The first word of the arguments to PWBMAIN contains a single character, but it is not a recognized one.

System Action: The client terminates.

27I Address structure *sub-1***.**

Explanation: This message is issued when TCP/IP reports errors on the `bind()` or `connect()` socket calls and there is no specific error message available.

System Action: The TCP/IP error information is displayed on the standard error output.

28E No endpoints found to monitor.

Explanation: The server did not find a statement containing an endpoint to listen to in its input file (PWB SERVER).

System Action: The server terminates.

30E Unrecognised command *sub-1***.**

Explanation: The partner sent a request for a function to be carried out, but the request code is not recognized.

User Response: Check if the client and the server are on compatible levels. They are supposed to do so during their initial handshake, but maybe the programmer overlooked something.

System Action: An error packet “invreq” is returned.

31E Insufficient storage to allocate *sub-1* **bytes.**

Explanation: The operating system returned a null pointer on a request to allocate storage.

User Response: Check whether the number is ridiculously large. If it is, the client may be connected to something other than the Plumbers’ Workbench. And some text could be interpreted as a packet of very large size.

System Action: Depends on the calling routine.

33E Unsigned integer argument expected; received *sub-1***.**

Explanation: An unsigned integer is expected for an argument to an external function in the Plumbers’ Workbench, but the argument was not processed completely by the `strtoul()` function.

User Response: Look for corrective service.

System Action: The function call is rejected; REXX signals a syntax error.

34E Environment variable *sub-1* **is not set.**

Explanation: An environment variable is required to specify a parameter and no default is assumed.

System Action: The client terminates.

35E Too many words in string *sub-1*.

Explanation: The string is being parsed into words, but it contains more words than needed.

System Action: Depends on the calling routine.

38I WinErr: *sub-1*.

Explanation: Additional detailed error information is available from the Presentation Manager.

39E PM error *sub-1* in call to *sub-2*.

Explanation: An unexpected error code is received on a call to the Presentation Manager. The decimal number is shown.

User Response: Refer to the description of the function to determine the meaning of the error code if there is no accompanying message 38. Check that you are running the correct level of OS/2.

System Action: Message 38 is issued if there is supplemental error information available from the Presentation Manager.

40E Unrecognised window type *sub-1*.

Explanation: The second argument to the external function PwbPMCreate is not recognized.

User Response: Check that the PWBMAIN.EXE module is compatible with the PWBCTL.CMD REXX program.

System Action: The function call is rejected; REXX signals a syntax error. The client terminates.

41E Unrecognised message type *sub-1*.

Explanation: The second argument to the external function PwbPMMessage is not recognized.

User Response: Check that the PWBMAIN.EXE module is compatible with the PWBCTL.CMD REXX program.

System Action: The function call is rejected; REXX signals a syntax error. The client terminates.

42E Error *sub-1* in call to *sub-2*.

Explanation: This is a catch-all message for unexpected errors. An error is detected in a call to the operating system. The return value and the name of the function are substituted.

User Response: Check the documentation for the function. Check that your OS/2 is at the required level.

System Action: Depends on the calling routine.

43E Incomplete length in packet (got *sub-1* bytes instead of *sub-2*).

Explanation: TCP/IP signalled end-of-file while the Plumbers' Workbench was reading the first four bytes of an incoming packet. These four bytes should contain the length of the entire packet. It is likely that the Plumbers' Workbench is not connected to a compatible server.

User Response: Check the TCP/IP connection and your setup. If you are sure that compatible client and servers are connected, you may have to resort to a TCP/IP trace to determine whether the packets transmitted on the network are correct or not.

System Action: The client terminates.

44E Incorrect length in packet *sub-1***.**

Explanation: TCP/IP signalled end-of-file prematurely while the Plumbers' Workbench was reading a data packet. The first four bytes contain the length of the entire packet, but this number of bytes was not received.

User Response: Check the TCP/IP connection and your setup. If you are sure that compatible client and servers are connected, you may have to resort to a TCP/IP trace to determine whether the data is correct.

System Action: The client terminates.

45E Unexpected handshake: *sub-1***.**

Explanation: The first packet received from the partner is not an "Iam" packet. The packet type is substituted.

User Response: Check that the partner is the correct server or client.

System Action: The client terminates. The server closes the connection.

46E Partner is too downlevel: *sub-1***.**

Explanation: An "Iam" packet is received from the partner. It indicates that the partner is running at a code level that is too low for the present Plumbers' Workbench.

User Response: Ensure that the client and the server are compatible.

System Action: The client terminates. The server closes the connection.

47I Partner terminates: *sub-1***.**

Explanation: The partner is about to terminate. On the server side, this message indicates that the client is about to close its connection to the server, which is quite normal. This message is not sent by the server.

48I Plumbers′ Workbench level *sub-1* **dated** *sub-2***.**

Explanation: A greeting message.

49E Unrecognised work item type *sub-1***.**

Explanation: An incoming item of work has a type that is not recognized. This is likely to be the result of a programming error in the Plumbers' Workbench.

System Action: The work item is ignored. This may lead to loss of communication with the partner.

50E Unrecognised packet type *sub-1***.**

Explanation: A packet is received for which the task was not prepared. This is likely to be the result of a programming error in the Plumbers' Workbench.

System Action: The packet is ignored. This may lead to loss of communication with the partner.

51E Unexpected packet type *sub-1* **in state** *sub-2* **process** *sub-3***.**

Explanation: A packet is received for which the task was not prepared. This is likely to be the result of a programming error in the Plumbers' Workbench.

System Action: The packet is ignored. This may lead to loss of communication with the partner.

52I Request from TCP port *sub-1* on host *sub-2* is not authorised.

Explanation: A connection request is rejected by the server.

User Response: Check for a hacker being afoot, or a frustrated plumber aloof.

System Action: The socket is closed.

53E Host name *sub-1* cannot be resolved (return code *sub-2*).

Explanation: The host name cannot be resolved to an IP address.

User Response: Check the name of the host and your nameserver setup. Check that the workstation is connected to the network.

System Action: The client terminates.

54E Host *sub-1* is not available.

Explanation: TCP/IP cannot reach the specified host system, but it could resolve its name to an IP address. (EADDRNOTAVAIL was returned.)

User Response: Check the condition of the network.

System Action: The client terminates.

55E Port *sub-1* on host *sub-2* is not active.

Explanation: No server is listening on the port. The host is up. (ECONNREFUSED was returned.)

User Response: Ensure that the server is running. Check that you have specified the correct remote port to the client.

System Action: The client terminates.

56E Network containing host *sub-1* is not reachable.

Explanation: TCP/IP cannot reach the network that contains the specified host. (ENETUNREACH was returned.)

User Response: Check your network.

System Action:

57E Connection attempt to host *sub-1* timed out.

Explanation: TCP/IP received no response from the host. (ETIMEDOUT was returned.)

User Response: Ensure that the system running the server is up and that its network connections are running.

System Action: The client terminates.

58E Partner detected error: *sub-1*.

Explanation: The partner sent an error packet. The error text is substituted.

System Action: The request is terminated.

59E Unrecognised command switch *sub-1*.

Explanation: A leading word begins with a slash (/), but it does not contain one of the supported command switches (options).

System Action: The client terminates.

60E Operand missing for command switch *sub-1***.**

Explanation: The last word of the arguments contains a command switch that should be followed by a value, but the argument string ends without such a word.

System Action: The client terminates.

61I Cannot erase file *sub-1***.**

Explanation: The file shown could not be erased. The file name can be incorrect or the path to the file may not exist. Or the file simply does not exist.

System Action: The Plumbers' Workbench ignores this error.

62E Code page *sub-1* **is not numeric.**

Explanation: The code page number shown is not a whole number.

System Action: The server terminates.

63I Code page *sub-1* **is not supported; using** *sub-2***.**

Explanation: The code page of the system is not supported for data interchange. Note that you should set the code page of the system to the code page of the data being transferred. If the data is REXX programs or C language programs, it is likely that you are best off using code page 1047 on the host side.

System Action: The default code page

64W Pipeline level 1.1.9 sublevel 33 (hex) required; *sub-1* **sublevel** *sub-2* **is active.**

Explanation: The *CMS Pipelines* module is not at the minimum fix level used for testing of the Plumbers' Workbench.

User Response: Look for a later version of *CMS Pipelines*.

If you are an IBM customer and an enthusiastic plumber, you might consider joining the *CMS Pipelines* field test program. You will then receive the latest code and documentation. But you will be running "unsupported" code.

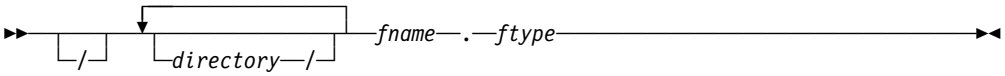
If you are an IBM employee, find out how to activate the VMTOOLS version of *CMS Pipelines*; it is likely to be installed on your system already.

System Action: Processing continues. TCP/IP communications may fail.

65E Incorrect file name format *sub-1***.**

Explanation: The file name is not in the standard portable format.

User Response: Ensure that the file name is of this form:

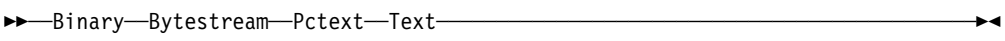


System Action: The request is terminated.

66E Incorrect file format *sub-1***.**

Explanation: The transmission type for download, pipe, or upload is not a supported one. Note that the transmission format is a required operand in the requests that are issued using the /LITERAL and /FILE options.

User Response: Ensure that the transmission format is one of these:



System Action: The request is terminated.

8.2 Messages Sorted by Text

Use the following list as a cross reference when only message text is known. Look up the message text, locate the corresponding message number, then match the message number to those listed in the previous section for the explanation, user response, and system action.

27I Address structure *sub-1*.
25E Authorities must be a single letter. *sub-1* characters found.
26E Authorities must be n, r, w, c, or a. *sub-1* found.
13E C Library error: *sub-1* (in *sub-2*).
61I Cannot erase file *sub-1*.
62E Code page *sub-1* is not numeric.
63I Code page *sub-1* is not supported; using *sub-2*.
57E Connection attempt to host *sub-1* timed out.
34E Environment variable *sub-1* is not set.
42E Error *sub-1* in call to *sub-2*.
2E Error running REXX program *sub-1*. Return code is *sub-2*.
22E ERRNO *sub-1* on socket call.
9E File *sub-1* not found.
54E Host *sub-1* is not available.
53E Host name *sub-1* cannot be resolved (return code *sub-2)*.
43E Incomplete length in packet (got *sub-1* bytes instead of *sub-2)*.
66E Incorrect file format *sub-1*.
65E Incorrect file name format *sub-1*.
44E Incorrect length in packet *sub-1*.
31E Insufficient storage to allocate *sub-1* bytes.
5E Integer argument expected; received *sub-1*.
15E Integer expected. *sub-1* was specified.
17E Integer is too large. *sub-1* was specified; the maximum is *sub-2*.
16E Integer is too small. *sub-1* was specified; the minimum is *sub-2*.
20E Internet address *sub-1* is not correct format.
21E Lost connection to *sub-1* port *sub-2*.
7I Message discarded for unused port *sub-1* from *sub-2* sequence *sub-3* cmd *sub-4*.
56E Network containing host *sub-1* is not reachable.
28E No endpoints found to monitor.
12E Novalue raised on line *sub-1* of *sub-2* for variable *sub-3*.
60E Operand missing for command switch *sub-1*.
58E Partner detected error: *sub-1*.
46E Partner is too downlevel: *sub-1*.
23E Partner terminated session prematurely.
47I Partner terminates: *sub-1*.
64W Pipeline level 1.1.9 sublevel 33 (hex) required; *sub-1* sublevel *sub-2* is active.
48I Plumbers' Workbench level *sub-1* dated *sub-2*.
55E Port *sub-1* on host *sub-2* is not active.
39E PM error *sub-1* in call to *sub-2*.
52I Request from TCP port *sub-1* on host *sub-2* is not authorised.
11E Syntax raised on line *sub-1* of *sub-2*.
3E Too few arguments in function call. *sub-1* received; *sub-2* expected.
24E Too few parameters. *sub-1* specified; *sub-2* are required.
8E Too many arguments in function call. *sub-1* received; *sub-2* expected.
35E Too many words in string *sub-1*.
19E Unable to initialise socket interface.
10E Unable to open file *sub-1*.
45E Unexpected handshake: *sub-1*.

51E Unexpected packet type *sub-1* in state *sub-2* process *sub-3*.
4E Unexpected REXX interface. Result string is *sub-1* bytes; *sub-2* are required.
30E Unrecognised command *sub-1*.
59E Unrecognised command switch *sub-1*.
6E Unrecognised command verb *sub-1*.
41E Unrecognised message type *sub-1*.
50E Unrecognised packet type *sub-1*.
40E Unrecognised window type *sub-1*.
49E Unrecognised work item type *sub-1*.
33E Unsigned integer argument expected; received *sub-1*.
1E Unsupported protocol *sub-1*.
38I WinErr: *sub-1*.

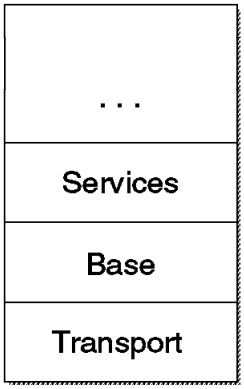
Part 2. Implementation

The second part of this document contains information about how we constructed the Plumbers' Workbench. It also discusses some of the experiences we gained in the process:

- Its design
- The systems we used to build it
- Notes about some books we read
- What we did to build it
- How we manage message repositories

Chapter 9. The Design of the Plumbers' Workbench

The Plumbers' Workbench is a workstation application that has CMS in its back room. Application functions, such as the VMPIPE command, the Presentation Manager engine, and the pipeline editor are implemented by a protocol stack, which is on top of a base layer. The base layer is an abstraction of network interfaces; below it is a protocol-dependent transport layer. That is, the specifics of the particular network interface are taken care of in the physical layer; the base layer relies on such generic functions as *send* and *receive*, which transmit packets between the two sides of the Plumbers' Workbench.



The workstation side is implemented as a REXX program that calls C language subroutines in the traditional way. The server is written as a data flow program using *CMS Pipelines*.

9.1 Physical Transport Layer

The physical transport layer between the client on the workstation and the server on CMS is implemented using TCP sockets, which provide a full duplex, connection-oriented byte stream between the client and the server. TCP/IP guarantees that bytes are delivered in the order they are sent.

The Plumbers' Workbench sends its packets through the TCP layer by prefixing them with a four-byte length field, which allows the other end to deblock the byte stream into packets.

The client side interfaces to TCP/IP through the C language interface routines. The server is built around the *CMS Pipelines* stages *tcplisten*, which listens for connection requests, and *tcpdata*, which is the client's agent in the host pipeline.

The steps in setting up a client/server connection, running a request, and terminating the connection are:

1. The server listens for connection requests on the specified port number. The *tcplisten* stage does this by creating a socket, binding it to the port, and listening.
2. A client issues a connect request. It also creates and binds a socket; it then issues the connect request to connect to the server.
3. When a connection request is processed by TCP/IP, the *tcplisten* stage accepts the request (it has to; this is the only way to discover the client's identity), offers the socket representing the request to the world by the givesocket function, and writes an output record that describes this request. When the record has been consumed, the *tcplisten* stage closes the socket.
4. The pipeline must now decide what to do about this connection request. And it must make its decision before it consumes *tcplisten*'s output record. If it consumes the record that represents the request without starting a *tcpdata* stage to service it, the *tcplisten* stage will close the socket and TCP/IP will

terminate the connection. The client then receives the ECONNRESET error number.

5. Normally, the pipeline would pass the record to a *tcpdata* stage that has been started to service the request. The *tcpdata* stage uses the takesocket function to get the socket that the *tcplisten* stage gave away. It then consumes the record.
6. The *tcplisten* stage closes the socket, which has already been taken, and prepares to listen for more connection requests.
7. The client and server exchange information about themselves by sending an `Iam` packet. These packets are sent simultaneously. Each side reads the other's identity packet and verifies that they are compatible. To allow for different levels, the onus is on the higher-level side to reject the lower-level side if it knows that it will be requiring a function that the partner cannot supply.
8. The client then sends a request packet to the server. Depending on the request, data packets may flow in one or both directions.
9. When the client has no more requests for the server, it sends an `end` packet and terminates. This closes the socket; TCP/IP dismantles the connection.

9.2 Base Protocol Layer

The base protocol layer is implemented as a REXX program. The client program is a "normal" program, whereas the server is implemented as a *CMS Pipelines* filter.

The two base layer programs are constructed from three REXX "source" files:

- The common code, which implements the protocol. It calls subroutines to interact with the external world and with its partner.
- OS/2-specific subroutines, which are included on the client side only. On OS/2, some calls go directly to the REXX I/O functions or to external REXX functions that are written in the C language.
- CMS-specific subroutines, which are included on the server side only. These functions issue *CMS Pipelines* requests and call CMS commands to provide the base layer abstraction.

The base layer implements subtasks, which are connected to sub-ports, also managed by the base layer.

REXX has no subtasking facilities; to emulate subtasking, the REXX program that implements the base layer protocol maintains a state variable for each sub-task. A central dispatching routine calls action routines whenever a packet is received or a record becomes available. The action routine then performs whatever action is required and returns to the central dispatcher. A sub-task is typically associated with a sub-port and a file handle, for example to send the file through the port. Data transfer is paced by acknowledgment packages from the partner. This prevents the network from being flooded when a large file is transmitted.

9.2.1 Communicating on the Base Layer

At the base layer, communication is symmetrical; either side can request a function of the other. Therefore, the other side is called the *partner*.

Base layer packets contain a prefix for routing between the subtasks, and optionally payload data. The prefix contains four blank-delimited words (in ASCII): the origin and destination sub-ports, a sequence counter, and the packet type.

Note that there is no central agency for assigning sub-ports to subtasks. Rather, the side that originates a request will eventually discover the sub-port that its partner has allocated to process the request.

To get things started, the requesting side sends a request packet to its partner's sub-port 0. This sub-port is reserved for request packets. The partner then allocates a sub-port to process the request. It knows the origin sub-port and can send an acknowledgment packet to that sub-port, which in turn informs the originator of the serving sub-port.

For example, the originating side may have allocated sub-port 3 to a particular sub-task. It then sends a `getf` packet to the partner to request the transmission of a file. The partner might allocate sub-port 17 for this request, verify the existence of the file and then transmit a `dats` packet to notify the originator of the file characteristics. When the originator receives a packet from the partner's sub-port 17, it knows that is should send its acknowledgment packet to that port.

9.3 The VMPIPE OS/2 Command

The VMPIPE command is a C language program. It starts by establishing the base layer connection with the server. Once the server link is established, the VMPIPE module installs the external functions (refer to Appendix A, "REXX External Functions" on page 73) and calls the REXX program PWBCTL.CMD to process the arguments and issue the resulting requests.

When all requests have been issued and all subtasks have terminated, the REXX program terminates, and VMPIPE closes the connection to the server and terminates.

9.4 The PWB EXEC Server

The server is implemented in the PWB EXEC. This program loads the PWB message repository and starts a multistream pipeline to process immediate commands and listen on the communications endpoint for connection requests. In normal operation, the server terminates only when the immediate command STOP is issued.

For TCP/IP, it uses *tcplisten* to wait for connection requests. The output from *tcplisten* is passed to the REXX filter *pwbtcpvalidate*, which verifies that the client is authorized to access the server and then adds a server thread to the pipeline set (using the ADDPIPE pipeline command). Multiple server threads run concurrently by nature of their being unconnected pipeline specifications in the same pipeline set.

The server thread is the REXX filter *pwbctl*. It receives packets from the *tcpdata* stage on its primary input stream; its primary output stream is passed to the input of the *tcpdata* stage, which in turn sends the packets back to the client.

A client request will, in general, cause the server thread to allocate a pair of input and output streams to the request and then add the appropriate pipeline segment using the ADDPIPE pipeline command. Thus, server requests can also run in parallel.

9.5 Running a Remote Pipeline

As an example of higher-level protocol stacks, consider the VMPIPE command running a remote pipeline. That is, the contents of the standard input stream are passed through a pipeline segment on the server and the resulting output is passed to the VMPIPE command's standard output.

The remote pipeline request allocates two sub-ports and two subtasks, because the data stream flowing to the server and the one flowing back are unrelated.

The first sub-task reads from the standard input and passes the data to the first sub-port; the second sub-task reads packets from the second sub-port and passes the data to the standard output. The `pipe` request packet is sent from the first sub-port; it contains the data format for transmission (binary or text), the number of the second sub-port, and the pipeline segment to run.

To process the request, the server then allocates two subtasks, two sub-ports, and a pair of input and output streams.

The server thread uses the ADDPIPE pipeline command to connect the pipeline segment between the output stream and the input stream that are allocated to the request. If the data format is ASCII text, stages are added to the beginning and to the end of this pipeline. The stages added to the beginning convert the byte stream into CMS records of the appropriate code page before they enter the pipeline stages specified by the client. The stages added to the end translate the output records and convert them back to a byte stream before they are returned to the workstation.

The server thread then enters its dispatching loop waiting for work, which will arrive as records on the input streams. A record on the primary input stream contains a packet from the client. The packet can contain a message for any of the active sub-ports.

If it is a data packet for the first sub-port, it contains a record, which is passed to the output stream that was allocated to the request. This in turn passes the record through the agent pipeline.

A record on the input stream allocated to the sub-task contains output data from the pipeline. The record is sent back to the client.

Chapter 10. Development and Testing Environment

The Plumbers' Workbench was written at the International Technical Support Organization in Poughkeepsie using an IBM PC Server 500 with 32 megabytes of storage and 2 gigabytes of disk space. The machine was connected to a VM system through TCP/IP over a token-ring network.

The system was running:

1. OS/2 Version 3 (Warp)
2. Communications Manger 1.11
3. KEDIT for Windows, which works fine under WIN-OS2
4. The IBM C Set ++ FirstStep compiler
5. C/C++ Tools Version 2.01
6. TCP/IP for OS/2
7. TCP/IP for OS/2 base kit
8. Developer Connection for OS/2

Chapter 11. Books We Have Read

This chapter lists some of the books we have read to be able to build the Plumbers' Workbench. The reviews are grouped into 11.1, "IBM Publications" and 11.2, "External Publications" on page 52.

Learning to program the Presentation Manager in a month means a lot of reading. Our over-all impression of the IBM manuals is that they deal well with the individual topic, but they lack information about the protocols, which are implemented by the sequence of events and window messages.

The Presentation Manager books are also available on the Developer Connection for OS/2 CDROM.

11.1 IBM Publications

We used the hardcopy version of the IBM publications described below.

- **OS/2 Warp, Version 3 Presentation Manager Programming Guide - The Basics**

  Document Number G25H-7103-00

  This book introduces windowing concepts and the simpler controls that are provided by the Presentation Manager.

  Each chapter is structured as a descriptive section, which is followed by reference material showing the syntax of related function calls and format of the relevant messages.

- **OS/2 Warp, Version 3 Presentation Manager Programming Guide Advanced Topics**

  Document Number G25H-7104-00

  This book is a logical follow-on to the basic guide; its deals with more complex controls.

- **OS/2 Warp, Version 3 Presentation Manager Programming Reference**

  Document Number G25H-7190-00 (Volume 1) and G25H-7191-00 (Volume 2)

  These two volumes contain the definitive reference for Presentation Manager programming. The first volume contains the description of the function calls; the second volume contains the description of the message processing performed by the public window classes.

  For two volumes of such size, the occasional error and inconsistency is quite rare.

- **IBM Transmission Control Protocol/Internet Protocol Version 2.0 for OS/2: Programmer's Reference**

  IBM form number SC31-6077-04

  We read only Chapter 2, which describes the socket Application Programming Interface. The description of the socket calls is reasonably good.

  The manual does not describe the interaction of sockets with threads and processes, except to say that each process must perform a `sock_init()` call to initialize its access to sockets. It turns out that the socket space is global.

Any process or thread can access any socket, be it opened by the process or not. It seems that programmers who use OS/2 TCP/IP should drive with due care and attention.

11.2 External Publications

We also read “real” books, which are reviewed below. The books are given a rating according to their usefulness to _; 1 is most useful.

- **Advanced OS/2 Presentation Manager Programming**

  By Thomas E. Burge and Joseph Celi, Jr. IBM Form Number SR28-4646; ISBN 0-471-59198-X

  This is a well-written and comprehensive book. It covers most Presentation Manager topics related to the management of windows. It does not attempt to explain the drawing primitives.

  The example programs are clear and you do get to see the windows so that you can relate the programming to the appearance on the workstation.

  Usefulness to _: 1.

- **The Art of OS/2 2.1 C Programming**

  By Kathleen Panov, Larry Salomon and Arthur Panov. IBM Form Number SR28-5357; ISBN 0-471-58802-4

  The title of this book may be a misnomer. It deals with Presentation Manager programming in the C language rather than with writing C language programs for OS/2 in general.

  This book contains many warnings about errors or omissions in the IBM manuals.

  This is a book written by programmers for programmers. It could have more “screen scrapes” showing what the windows look like.

  Usefulness to _: 3.

- **OS/2 Presentation Manager Programming Hints and Tips**

  By Bryan Goodyer. IBM Form Number SR28-5598; ISBN 0-07-707776-8

  This is a cookbook. It contains the answers to many “how to” questions.

  Usefulness to _: 1.

Chapter 12. Tips and Advice on Developing Client/Server Applications for OS2 and CMS

12.1 Choose Your Home

When you are working on a project that requires both OS/2 and CMS files and programs, it may be more comfortable to keep the source for all programs on one of the two systems and then download (or upload) copies for compilation or execution.

We have done previous AIX projects using XEDIT and C/370 to edit and syntax check our code before downloading it and building the executable file on AIX.

In the Plumbers' Workbench project, we have taken the opposite approach. All code is on the workstation; the REXX executables are built under OS/2 and then uploaded to CMS.

12.2 Find an Editor—Our Experiences with KEDIT

If you are programming on OS/2 or on CMS, you will be spending most of your time interacting with a text editor; the choice of editor can influence your productivity significantly.

Your choice is likely to be a descendant of CMS' XEDIT or a descendant of GNU's EMACS editor. Both are highly customizable and both support a macro language.

We expect that you have experience with XEDIT and with writing XEDIT macros and that you will be more comfortable in an XEDIT-like environment than you would be learning a new editor, albeit a very powerful one.

We have been using KEDIT for OS/2 for some years now. It is in general compatible with XEDIT. When it is incompatible, it is so because the revised behavior is markedly superior to XEDIT's. Two of its extensions over XEDIT are:

- The user has complete control over how the editor should react, because all keyboard and mouse events invoke macros, which can be replaced.
- It provides an "undo" facility, which allows the user to revert the file to a previous state. You can undo progressively, even past the last save of a file. You can undo your undos (called redo). Undo is addictive.

We have extended many of our XEDIT macros to be bilingual so that they work both under CMS and under OS/2. Some macros are even trilingual; they also run with PDF under TSO.

We installed both KEDIT for Windows and its precursor, KEDIT for OS/2.

KEDIT for Windows has many enhancements over KEDIT for OS/2. We particularly like:

- Its CUA look-and-feel. Each file is shown in a separate window on KEDIT for Windows' desktop. Windows can overlap and be partially visible. File windows can be minimized to this desktop.
- Its syntax coloring of programs. KEDIT for Windows provides a parser for REXX syntax and one for C language syntax. The nesting of control structures and parenthesized expressions is highlighted in different colors. Keywords, literal strings, and variables are also shown in different colors. We find this extremely helpful.
- Its dialog to control setable options. This menu allows many options to be set permanently, which means fewer commands in the profile.
- Its clipboard support. While the OS/2 version supported the clipboard by nature of running within the CMD.COM window, KEDIT for Windows supports the clipboard natively. You can clip KEDIT selections to the clipboard and paste at the cursor position.

But there are some drawbacks when compared to the native OS/2 KEDIT:

- KEDIT macros are written in a subset of REXX, known as KEXX; OS/2's REXX is not supported.
- A KEXX macro must not begin with a REXX comment. Thus, it is not possible to use the same file with both KEXX and REXX.
- Commands are issued to the DOS box. This means that you cannot invoke OS/2 utilities from KEXX macros.

We found that the ease-of-use of KEDIT for Windows outweighed the annoyance of writing a KEDIT macro to convert our macros from REXX to KEXX.

KEDIT can be ordered by contacting:

Mansfield Software Group, Inc.
P.O. Box 532
Storrs, CT 06268
USA

12.3 Install a C Compiler

We installed the IBM C Set ++ FirstStep compiler from a CDROM. The installation was "uneventful". The part number for the First Step for OS/2 C Set ++ CD-ROM pack is 82G-3747.

12.4 Order the TCP/IP Toolkit

You need the C language header files for TCP/IP to be able to write programs that use TCP/IP services. The toolkit is available by ordering product number 65G-1220.

12.5 Join the Developer Connection for OS/2

The Developer Connection (DevCon) is a subscription service that comes out four times a year; each issue contains a CD-ROM, a newsletter, and occasionally other materials. Fixes to DevCon are integrated into each new issue.

12.5.1 IBM Customers

In the following sections, we describe how IBM customers may obtain the Developer Connection for OS/2.

12.5.1.1 In the US

The Developer Connection for OS/2 can be ordered directly from IBM. In the United States, customers can obtain DevCon by calling 1-800-6DEVCON (1-800-633-8266).

12.5.1.2 Outside the US

Other geographies may have different distribution/pricing policies, so the “local” number should be called for ordering information.

Canada 1-800-561-5293

Asia/Pacific 61-2-354-7684

Mexico 627-2444 (Mexico City) 91-800-00639 (Country)

Brazil 0800-111205

In addition, DevCon can be ordered direct from IBM SPC in Denmark if the customer is outside one of the above geographical areas. After dialing the international access code, the customer should call the appropriate number to speak with an operator speaking one of the following languages:

English 45 +48101500

Dutch 45 +48101400

French 45 +48101200

German 45 +48101000

Italian 45 +48101600

Spanish 45 +48101100

Note that 45 is the country code for Denmark.

12.5.2 Employees of IBM

Documentation and additional materials for DevCon can be found at this FTP site:

```
ftp devcon.bocaraton.ibm.com
```

We tried to download the PostScript source files for the manuals from the Developer Connection for OS/2 FTP site. They trickled in at about 10 kilobytes per second. We then defined an 800-cylinder temporary minidisk and ran PS370 to format them into LIST3820 files, which we could print duplexed on an IBM 3827 printer.

Don’t emulate this mistake. Get the Developer Connection for OS/2 CDROMs immediately. The reference material is easier to retrieve online, because the

hardcopy reference books do not mention the function being described in the running title.

12.6 The OS/2 Toolkit

CMS users will probably be surprised to learn that OS/2 is shipped without C language header files for its Application Programming Interface. Header files and library stub routines are shipped as part of the toolkit, as are most of the system utilities to generate modules and so on. (Imagine that the CMS GENMOD command was shipped as part of some other product.)

12.6.1 Order the OS/2 Toolkit

Finding the correct level of the toolkit can be a bit of an art. Because it is clearly very important, it is shipped with the IBM C Set++ compiler (but that level is rather old), as product number 82G-3733 (which is not quite so old), and on the Developer Connection for OS/2's FTP site. The latter is the one you want.

But installing it is slightly convoluted:

1. Download the eighteen diskette images from the FTP site.
2. Find a utility that will write the diskette images to diskettes. We used EPM, which is an IBM tool.
3. Run the installation EXE, which is on the first diskette.

We tried to unpack the diskette images directly to a directory on the hard disk and then install from there, but it seems as if the installation procedure insists on reading diskettes. It would also appear that all diskettes contain a file of a "known" name, which guides the program to the next diskette; this might be the real reason that our attempts at being smart were thwarted.

Having installed the toolkit, you need to add the libraries to the appropriate environment variables in CONFIG.SYS and then shut down the system and start it again, because a change to the LIBPATH takes effect only when the system is booted.

12.6.2 Using the OS2 Toolkit

In addition to the header files for calls to the system libraries, the toolkit contains several utilities that are crucial to generating a working program module file or a dynamic link library. You will definitely need to understand how to use:

- The NMAKE program, which builds an up-to-date executable from its parts, compiling and running utilities as needed to react to updated source files.
- The RC program, which compiles a resource definition file into objects that you can refer to, such as menus, icons, string tables, and more. The resource compiler also attaches the compiled resource definitions to an executable module file.
- The LINK386 program, which creates a 32-bit executable from its constituent object parts.
- The MKMSGF program, which builds a message object file from a message repository.
- The MSGBIND program, which attaches a message object file to an executable module file.

In general, the documentation for the utilities in the toolkit is not quite at the same standard as the documentation for the base OS/2 operating system. And some of the utilities are definitely idiosyncratic. In some cases, not even a single additional blank will be tolerated.

12.6.3 The NMAKE Utility

The NMAKE utility is very much like the UNIX make command. It supports many enhancements over the standard make command, many of which seem to be inspired by the GNU make command. But there are pitfalls. In particular, be aware of these points:

- The documentation does not mention comments at all. But it appears that lines that have a pound sign (#) in the first position are treated as comments.

  ```
  #This is a comment
  main.exe:    Main.obj
  #This is also a comment
      #This is issued as a command, which fails.
      icc main.c
  ```

  Unlike other makes, you cannot indent a comment or put a comment on the same line as a command.

- The documentation states that a backslash (\) at the end of a line is used to continue a specification on the following line (this is known as *line splicing*).

  ```
  modules=main.obj utility.obj \
          rexxif.obj
  ```

  But read on:

- Comments appear to be removed before line splicing. This will not do what we expect:

  ```
  mac1=This is a continued macro\
  # But this is not the continuation
  mac2=And this is not a new macro; it is appended to mac1
  ```

  (Assignments, such as these, are called *macros* in nmake-speak.)

- At one point, the documentation mentions that only when the macro is defined at the command line can it be enclosed within quotes. This appears to be correct. The examples a few panels later of macros being enclosed in double quotes (either the entire macro or the right-hand side of the assignment) are plain wrong.

- There does not appear to be an escape mechanism for the backslash. It seems to be impossible to create a macro that has a trailing backslash. Thus, it appears not to be possible to set a macro to a directory path that ends with a backslash.

For an excellent description of a make command, as well as implementation issues, see the documentation for the GNU make.

12.7 What REXX Had in Store for Us

Though we have been using REXX since its inception at the beginning of the nineteen-eighties, the language still had two surprises. And the OS/2 REXX interpreter gave us quite a fright and then a slight scare.

We already knew that the OS/2 REXX documentation is not the most limpid.

12.7.1 The Signal Instruction

Case is respected in the value clause of a Signal instruction, but all labels in a program are translated to uppercase. This will raise a label-not-found error on all OS/2 interpreters and on CMS interpreters on current releases:

```
signal value "ProcessWrite"
⋮
ProcessWrite:   /* Write the result */
```

This will work:

```
signal value translate("ProcessWrite")
⋮
ProcessWrite:   /* Write the result */
```

12.7.2 Dropping Compound Variables

When dropping a compound variable, the variable becomes unset; it does not revert to the default it had before it was set. This will raise novalue:

```
Signal on novalue                   /* Trap bad references        */
port.=0                             /* Default of not in use      */
⋮
port.task=7                         /* Use this port              */
⋮
drop port.task
⋮
if port.task=0
   then call idletask
```

12.7.3 OS2 REXX Truncates at //

When the REXX interpreter is called for a command (as opposed to a function or a subroutine), it scans the argument string for two consecutive slashes. These and the remainder of the string are quietly deleted, even if they do not contain a valid interpreter option.

There is no antidote to this when invoking a REXX command directly from an OS/2 window (through CMD.EXE). Fortunately, the Plumbers' Workbench invokes a C language program first; it was changed to call the REXX program as a subroutine and then REXX does not scan for the double slash.

12.7.4 charin() Truncates Standard Input

The charin() function reads large blocks of a file without problems, but when attempting to read more than 512 bytes at a time from the standard input, it supplies only the first 512 bytes and then causes chars() to return 0. No doubt, this is related to the size of the OS/2 pipeline buffer.

We had to code like this to circumvent this problem:

```
PwbCharin:
parse arg stream, task
call PwbLog 5, 'Reading from' stream
bytes=word('32000 512', 1+(stream=''))/* read only 512 from stdin    */
data.task=charin(stream, , bytes)
```

Consensus is that this is a bug in charin().

Chapter 13. Managing Documentation, Help, and Message Repositories

The documentation for a CMS application is typically written using BookMaster markup, which is formatted into a document that can be printed and into an online book, which is browsed using BookMaster. The message repository is generated from the BookMaster source file using *CMS Pipelines* to generate the input for the GENMSG CMS utility, which creates the object message repository.

The *message prefix* is the first word of a message. It identifies the source of the message and shows its number. The structure of the message prefix depends on the operating system.

For OS/2, the message prefix contains the three-letter component identifier, the four-digit message number, the one-character severity code, and a colon.

For CMS, the message prefix contains the three-letter component identifier, the three-letter module name, the three-digit or four-digit message number, and the one-character severity code.

13.1 Message Repositories

In a distributed project, you will need to maintain message repositories both on the workstation and on the host. Administration and documentation are simplified if you use a common repository. This is possible, because both CMS and OS/2 provide message repositories and functions to extract message text from the repository and substitute message parameters into the message text.

Starting from the BookMaster source file, the Plumbers' Workbench message repositories were built by:

1. Transforming the BookMaster message tags into an OS/2 "input message file," from which the MKMSGF utility generates the object message file.
2. Transforming the OS/2 message file into a CMS repository file.

There are many commonalities between OS/2 message files and CMS message repositories. In both implementations, the aim is to remove language dependencies from the program code. They both support component identifiers, severity codes, and substitution of variables into the constant message text. But there are many subtle and not so subtle differences. The OS/2 facilities are a proper subset of the CMS ones; thus, it is possible to generate a CMS message repository mechanically from an OS/2 one, but the inverse operation is not possible in general.

Messages on OS/2 are associated with a three-character component identifier; the message number is always four digits. The module issuing the message is not identified, unless it is added explicitly to the message. OS/2 deletes the entire message prefix from informational messages; and it removes the severity code from other messages.

In addition to the features supported by OS/2 messages, CMS supports message formats, multiline messages, and the ability to substitute constants. A multiformat message is in essence several messages, from which one is

selected based on the format required at the time the message is issued. Multiline messages implement hard line breaks. That is, the message is unconditionally written as several output lines.

13.1.1 Generating Repositories from BookManager

The example in Figure 20 shows a few lines from the Script file that contains the message descriptions, from which the message repositories are built.

```
:msgl tsize=5 style=rule msgnosel=nodup.
:msgno.1E
:msg.Unsupported protocol :mv.sub-1:emv..
:xpl.The second argument word to
:hp4.PWBMAIN.EXE:ehp4. specifies the type of communications
protocol to use to connect to the server.
It is not recognized as a valid type.
:sysact.The client terminates.
:emsgl.
```

Figure 20. Sample BookMaster Message Definition

The makefile contains the lines shown in Figure 21 to regenerate the message repositories whenever the source file changes. The input file is 4523MSGS.SCR in the SCRIPT subdirectory of the directory where the makefile resides.

```
pipewb.txt:     script\4523msgs.scr
                vmpipe <$? >$@ "book2os2 | > pipewb txt p"

pipewb.msg:     pipewb.txt pipewb.mgb
                mkmsgf pipewb.txt $@ /v /l 9,1
```

Figure 21. Makefile Sections to Generate Message Repositories

The figure above contains two *description blocks*. The first one generates the input file to the message utility; the second one runs the message file utility to create the object file, which is included in the executable module.

In the first description block, the VMPIPE command is run when the source file is younger than the target file. It runs a REXX filter and then saves the output stream as a file (in EBCDIC) on file mode P. The first two words of the arguments specify redirection of the standard input and the standard output of the VMPIPE command. The remainder of the argument string is the pipeline to run on CMS. <$? is nmake-speak for read the source file (script\4523msgs.scr); >$@ means write the target file (pipewb.txt).

The second descriptor block invokes the OS/2 message utility to generate the object message file.

The REXX filter shown in Figure 22 on page 63 extracts the message tags from the SCRIPT source file and builds an OS/2 input message file, albeit in EBCDIC. (The file is converted to ASCII when it is transmitted back to the client.)

To do this, the REXX filter issues two subroutine pipelines. The first one determines the range of message numbers; the second one does the actual work. The range of message numbers must be known, because the OS/2 message utility requires that all messages be numbered consecutively. Neither

CMS nor BookMaster has such a requirement; the plumbing adds the required dummy messages. The message lines are stored in a stemmed array in the first subroutine pipeline and retrieved from this array in the second one. We could have written this as a single subroutine pipeline, but it would have been quite convoluted and partially built dynamically.

```
   /* Meat of the bookie to OS2 message file.                          */
   /*                                 John Hartmann  4 Aug 1995 12:07:28 */
   /*%test: pipe < 4523msgs script|book2os2 pwb|brw                    */

   Signal on novalue

   parse upper arg prefix .
   If prefix=''
      Then prefix='PWB'
   parse source s

   'callpipe (end \ name BOOK2OS2.REXX:4)',
      '\*:',
      '|strfind /:msg/',              /* Pare down to interesting lines */
1     '|stem msgs.',                    /* Save for later              */
      '|find :msgno.',                  /* Get message number tags     */
2     '|not chop after .',              /* Delete the message tag      */
3     '|spec 1;-2',                     /* Isolate the message number  */
      '|f:take first',                  /* Take the first one          */
      '|var first',                     /* Remember it                 */
      '\f:',                            /* Second and further          */
      '|take last',                     /* Just the last one           */
      '|var last'                       /* Save last message number    */
```

Figure 22 (Part 1 of 2). BOOK2OS2 REXX Converts a BookManager Script File to an OS/2 Message File

```
  'callpipe (end \ name BOOK2OS2)',
    '\stem msgs.',                        /* Get complete file          */
4   '|no:find :msgno.',                   /* Get message number tags    */
    '|not chop after .',                  /* Delete the message tag     */
5   '|spec pad 0',      /* Reformat the message number to fixed-format */
      '/'prefix'/ 1',                     /* Module prefix              */
      '1-* 4.5 right',                    /* Message number+sev in 4-8  */
      '/: / next',                        /* Colon and blank in 9-10    */
6   '|j: juxtapose',                    /* Prefix number to message text */
    '|spec w1 1 /./ nextw 1-* next',   /* Double-up the message number */
7   '|m: merge 1.7',                    /* Merge placeholders in      */
    '|unique 1.7 first',          /* Retain real message if it is there */
    '|literal' prefix,                  /* The prefix by itself       */
    '|literal ;Generated by' s date() time(),  /* Identify its origin */
    '|strip',                           /* Delete trailing blanks     */
    '|> pipewb txt i',                  /* Save for CMS build.        */
    '|*:',
    '\no:',                             /* Message lines here         */
    '|strfind /:msg./',                 /* Look for message tags      */
    '|not chop after .',                /* Delete the tag             */
8   '|change /:mv.sub-/%/',             /* Change substitutions       */
    '|change /:emv.//',                 /* ...                        */
    '|j:',                              /* Go get prefixed with number */
    '\literal',                         /* A line                     */
9   '|dup' last-first-2,                /* As many as we need         */
    '|spec pad 0',                     /* Generate default placholders */
      '/'prefix'/ 1',                   /* Module prefix              */
      'number from' first+1 '4.4 right',              /* The number */
      '/?:/ next',                      /* And the ?                  */
    '|m:'                               /* Merge with real messages   */

  exit RC
```

Figure 22 (Part 2 of 2). BOOK2OS2 REXX Converts a BookManager Script File to an OS/2 Message File

1 Save the lines that contain tags beginning with :msg in an array of REXX variables.

2 Delete the text from the beginning of each line up to and including the first period. The output line contains the message number and the severity code (1E).

3 Select from the first character to the second-to-last. In effect, this drops the severity code and leaves just the number. The first number is stored into one variable and the last number is stored into another variable.

4 Select the lines that contain the message number. The balance of the file is written to the secondary output stream, which is processed later in the pipeline. The next stage deletes the tag.

5 Construct the OS/2 version of the message prefix:

1. Pad short fields with the number zero.
2. Put the component ID as a constant in columns 1-3.
3. Insert the message number and the severity code right-aligned in a five-byte field. This pads shorter message numbers with leading zeros, because the pad character has been set to a zero.
4. Append a colon and a blank.

Thus, the output record contains the message prefix and a trailing blank.

6 Prefix the message prefix to the message text, which is generated later in the pipeline. The next stage doubles up the prefix and adds a leading period to the copy of the prefix, which becomes the first word of the message text.

7 Merge message lines with a supply of dummy messages. The merge key does not include the severity code. Thus, for messages that are in the source file, the real message will precede the dummy message. (Such are the rules for the *merge* gateway.) The following stage selects the first occurrence of a message. This discards the dummy message when there is a real message.

8 Process the message text line by dropping the tag and then changing the substitution tags to a percent sign, which is the character required by the OS/2 utility.

9 Create as many null records as required to generate the dummy messages. Then make the dummy messages in the following stage.

The pipeline in Figure 23 creates the input to the CMS message utility from the input to the OS/2 message utility.

```
/* Build a CMS Message Repository from an OS2 one.                  */
/*                                 John Hartmann  20 Jun 1995 16:00:23 */
Signal on novalue

parse arg fn .
```

Figure 23 (Part 1 of 2). OS2MSGCMS EXEC Converts an OS/2 Message Repository to CMS Format

```
   Address COMMAND
   'PIPE (end \ name OS2MSGCMS)',
      '\<' fn 'txt',                        /* Read OS2 message text file  */
      '|strnfind /;/',                      /* Delete comments             */
      '|strip',                    /* Remove trailing and leading blanks */
      '|locate 1',                          /* Delete blank lines          */
1     '|nlocate 8 /?/',                     /* Remove reserved messages    */
      '|t1: take 1',                      /* Get the component identifier */
      '|var langid',                        /* Set variable for CMS name   */
      '\t1:',                               /* Message lines here          */
      '|spec 12-*',                         /* Drop the duplicate prefix   */
      '|spec 4.4 1 /0101/ n 8.1 n 10-* n',/* Make nnnnfflls from prefix */
2     '|chop: chop 10',                     /* Split into prefix and text  */
3     '|j:juxtapose',                  /* Append all lines for a message */
      '|literal % 4',      /* Four-digit numbers and % for substitution */
      '|pad 80',                            /* Make lines 80 byte cards    */
      '|> pwbume repos a fix 80',           /* Write the file              */
      '\chop:',                             /* The message text comes here */
4     '|spill 60', /* Chop it into lines that are at most 60 characters */
      '|j:'                                 /* Prefix them with the number */

   fn=langid'UME'                           /* Get file name               */
   'GENMSG' fn 'REPOS A' langid
```

Figure 23 (Part 2 of 2). OS2MSGCMS EXEC Converts an OS/2 Message Repository to CMS Format

1 OS/2 requires that messages be numbered consecutively and that all messages be present within the range of numbers being used. A placeholder message is specified by a question mark instead of the severity code.

2 *chop* writes the first ten characters of the input record to the primary output stream. This includes a trailing blank. The remainder of the record is then written to the secondary output stream.

3 The *juxtapose* stage prefixes the message prefix to each line of message text. It does this by reading the message prefix from its primary input stream into a buffer. Each time it reads a record from its secondary input stream, it prefixes the contents of the buffer to the record just read and then writes the composite record to its primary output stream.

4 *spill* splits records that are longer than sixty bytes into shorter records. Records are split between words. Thus, *spill* can produce more than a single output record from an input record. The output records are fed to the secondary input stream of *juxtapose*.

13.1.2 Giving OS2 Messages a CMS Flavor

To overcome the limitations of the way OS/2 messages are presented to the application, the Plumbers' Workbench stores the message prefix as the first word of the message text. When the message is retrieved, it will have part of the message prefix (the severity code is deleted) unless the message is an informational one, which is presented without any part of the message prefix. To allow for a quick decision between the two formats, the second occurrence of the message prefix is prefixed with a period. Figure 24 on page 67 shows a few lines of the Plumbers' Workbench input message file.

```
;Generated by CMS COMMAND BOOK2OS2 EXEC H1 BOOK2OS2 CMS 19 Jul 1995 18:54:43
PWB
PWB0001E: .PWB0001E: Unsupported protocol %1.
PWB0002E: .PWB0002E: Error running REXX program %1. Return code %2.
```

Figure 24. Sample OS/2 Input Message File

The code fragment to extract a message is shown in Figure 25.

```
/*********************************************************************/
/* Error Messages.                                                   */
/*                                                                   */
/* Retrieve   the   message   and   have  it  substituted.   Because */
/* informational messages do not return their message prefix and the */
/* severity  code is deleted from warnings and errors, we insert the */
/* message identifier into the message itself as the first word.  To */
/* distinguish  between errors and informationals, a dot is prefixed */
/* to  the  first  word.   If the retreived message has a dot in the */
/* first position, it means that the message was informational.      */
/*                                                                   */
/* Return the pointer to the first character of the message.         */
/*********************************************************************/

static char * PwbBuildMessage(
         char * msgbuf,               /* Message returned here       */
         long int buf_size,           /* Size of this buffer         */
         long int msg_num,            /* Message number              */
         char ** subs,                /* Substitution strings        */
         long int num_subs)           /* Number of substitutions.    */
{
   long int rv;
   unsigned long int msg_length;

   rv=DosGetMessage( subs,            /* Array of string pointers    */
                     num_subs,        /* Number of substitutions     */
                     msgbuf,          /* Buffer to receive message   */
                     buf_size,        /* Length of buffer            */
                     Pwb_MSG_BASE+msg_num,         /* Message number */
                     Pwb_MSG_FILE,    /* File containing message     */
                     &msg_length); /* Var to receive length stored   */

   if (rv)
   {
      fprintf(stderr, "Return code %d retrieving message:\n%s\n",
                       rv, msgbuf);
      exit(127);                      /* Give up                    */
   }
   return (MSG_DOT==*msgbuf) ? msgbuf+1 : msgbuf+10;
}
```

Figure 25. Retrieving Message with Substitution

Figure 26 on page 68 shows how to display a suitable dialog box based on the character string returned in Figure 25.

```
/*********************************************************************/
/* Display a message box for an error message.                       */
/*********************************************************************/

void PwbPMErrorBox(char * msg)
{
   int sw=MB_MOVEABLE;
   char title[80];
   int id;

   if (NULLHANDLE==hab) return;       /* Nocando                    */

   switch (msg[7])
   {
      case 'I':
         sw|=MB_OK | MB_INFORMATION;
         id=IDS_Information;
         break;
      case 'W':
         sw|=MB_OK | MB_ICONHAND;
         id=IDS_Warning;
         break;
      case 'E':
      default:
         sw|=MB_CANCEL | MB_ERROR;
         id=IDS_Error;
         break;
   }

   WinLoadString(hab, NULLHANDLE, id, sizeof(title), title);
   WinMessageBox(ancestor, ancestor, msg, title, 0, sw);
   return;
}
```

Figure 26. Displaying an Appropriate Message Box

OS/2 does not store constants for substitution in the message repository; rather, such strings are stored in the STRINGTABLE section of the module's resource file. In this example, the title of the window to display the message is set to show the severity code associated with the message. Figure 27 shows part of the Plumbers' Workbench's resource file:

```
#include "pwbpmrc.h"

STRINGTABLE
{
        IDS_Information "Information:"
        IDS_Warning     "Warning:"
        IDS_Error       "Error:"
}
```

Figure 27. Defining String Resources

The string identifiers are just numbers, but it is customary to define these numbers with macros. Figure 28 on page 69 shows part of the contents of the header file, which is included both in the resource file and in the source program.

```
#define IDS_Information 1001
#define IDS_Warning     1002
#define IDS_Error       1003
```

Figure 28. PWBPNRC.H Contains Macros to Define String Numbers

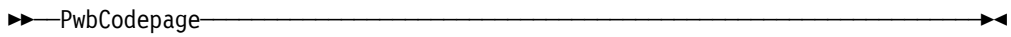
Part 3. Appendixes

Appendix A. REXX External Functions

This appendix describes the REXX external functions that are part of the Plumbers' Workbench.

A.1 PwbCodepage

Purpose: To return the current code page.



Arguments: None.

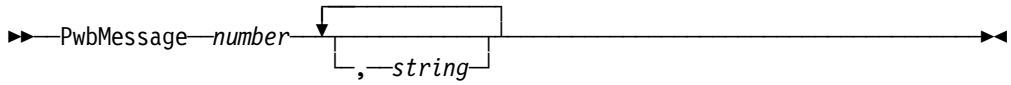
Return Value: The current code page number.

Example:

```
codepage=PwbCodepage()
```

A.2 PwbMessage

Purpose: To issue a message at the local system.



Arguments:

1. The message number of the message to be issued. Refer to Chapter 8, "Messages" on page 33 for a list of defined messages.
2. The first substitution string.
3. The second substitution string. Up to ten substitutions are supported by the underlying interface.

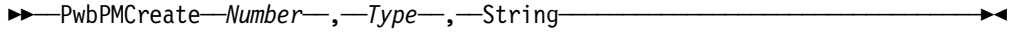
Return Value: The message number.

Example:

```
call PwbMessage 48, mylevel, mydate
```

A.3 PwbPMCreate

Purpose: To create a Presentation Manager window.



Arguments:

1. The sub-task number associated with the window. Events from the window are queued for this task.
2. The type of window to be created.

3. An argument string. The format depends on the type of window that is being created.

Return Value: A handle to the window that was created. NULLHANDLE (a zero) if no window was created.

Example:

```
bwnd=PwbPMCreate(17, "browse", "Browsing a File")
```

A.4 PwbPMDestroy

Purpose: To destroy a window.

Arguments:

1. The handle for the window to be destroyed.

Return Value: The return value from WinDestroyWindow.

Example:

```
call PwbPMDestroy bwnd
```

A.5 PwbPMMessage

Purpose: To send one or more messages to a window.

Arguments:

1. The handle for the window that is to receive a message.
2. A keyword that defines the function to perform.
3. A string, which depends on the particular type of function specified.

Return Value: Depends on the function specified. In general, a return value of zero indicates successful completion of the request.

Example:

```
rv=PwbPMMessage(bwnd, 'mle_put', data)
```

A.6 PwbSend

Purpose: To send a packet to the partner.

Arguments:

1. The partner's destination sub-port.
2. The sub-port for sending the message.

3. The message sequence number for the particular task.
4. The packet type.
5. The packet data (binary). The contents of the data part of the packet depend on the packet type.

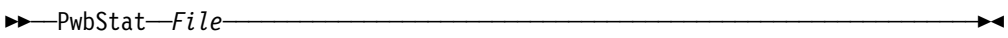
Return Value: Zero if the packet was sent successfully. This may indicate that the packet was queued at the transmitting protocol layer. A non-zero value indicates an error in sending the packet.

Example:

```
call PwbSend 0, 0, 0, 'getf', 'pctext profile.exec'
```

A.7 PwbStat

Purpose: To return the time stamps and size of a file.



Arguments:

1. The file name of the file to be queried.

Return Value: The return value is a null string when the file does not exist. When the file exists, the return value is an ASCII string, which contains eight blank-delimited words:

1. The time of last modification to the file.
2. The time of last access to the file.
3. The time of last control change to the file.
4. The time of creation of the file. OS/2 does not supply this information; a zero is returned.
5. The format of the file:

   b A normal file
   c A character special file
   d A directory

6. A zero as the placeholder for the logical record length.
7. A zero as the placeholder for the number of records.
8. The file size in bytes.

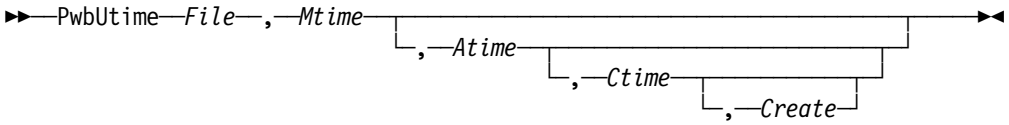
The time values are seconds since Midnight before January 1, 1970, Greenwich Mean Time.

Example:

```
stat=PwbStat(file)
if stat=''
   then say 'File' file 'does not exist.'
   else parse var stat mtime atime ctime . format . . size .
```

A.8 PwbUtime

Purpose: To set the time stamp for a file. This would normally be a file that was received from the partner.



Arguments:

1. The file name of the file to be modified.
2. The time of last modification to the file.
3. The time of last access to the file. This operand is optional; the time of last modification is used if this operand is omitted.
4. The time of last control change to the file. This operand is ignored.
5. The time of creation of the file. This operand is ignored.

The time values are seconds since Midnight before January 1, 1970, Greenwich Mean Time.

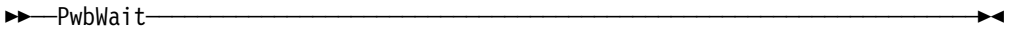
Return Value: The return value from the underlying library function.

Example:

```
call PwbUtime file, mtime, atime
```

A.9 PwbWait

Purpose: To wait for work. The thread's message queue is serviced. Control returns when one or more requests have been queued to the stemmed array `todo.`



Arguments: None.

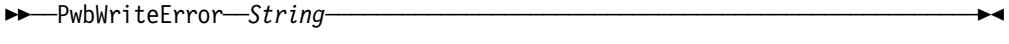
Return Value: Zero if work has been queued. One if the connection to the partner has been severed.

Example:

```
call PwbWait
```

A.10 PwbWriteError

Purpose: To write a line to the error log. The error log is written to the file specified in the environment variable `PWB_MESSAGEFILE`. The error log is written to the standard error if the environment variable is not set.



Arguments:

1. The line to append to the error log.

Return Value: Zero.

Example:

```
call PwbWriteError 'PWB Starting' date() time()'.'
```

Appendix B. Protocols

This appendix defines the protocol used for communication between the Plumbers' Workbench client and server.

B.1 Protocol Concepts

The client runs in the workstation; the server runs under CMS. Multiple clients can connect to the CMS server concurrently; these clients can run in the same workstation or in different workstations.

Each client establishes a single full-duplex connection to the server using a lower-level transport protocol. When using transport protocols that are half-duplex, such as APPC, two transport-layer sessions are established where each one flows packets in one direction only.

The server issues the ADDPIPE pipeline command to spawn an unconnected pipeline for each client that connects to it.

The protocol between the client and the server thread is symmetrical; each side can request services of the other one. The word *partner* is used to describe the other side of the connection. Both partners maintain a pool of sub-ports, which are used to identify individual sources and destinations of data.

The sub-port 0 (zero) is the only pre-assigned one. It represents the control port, to which the other side directs requests.

To support multiple concurrent data streams flowing between the two partners, sub-session tasks are established dynamically as required. A sub-session is represented by at least one pair of sub-ports. A particular sub-port is part of at most one sub-session task at a time.

B.1.1 Protocol Example

A typical request could be issuing a command on the server and transferring the response back to the client, where it could be displayed somehow. The transfer is done in these steps:

1. The client allocates a sub-port to represent the receiving end of the data stream that contains the command response.
2. The client sends a packet from this sub-port to the server's control sub-port requesting that a command be issued and the response be returned. The origin sub-port of this packet implicitly defines the destination sub-port to use for responses.
3. The server verifies that the client is authorized to issue commands. It rejects the request by sending a rejection message to the client sub-port.
4. The server allocates a sub-port to represent the command's response.
5. The server issues the command and transmits the response from this sub-port to the sub-port allocated by the client for this purpose.
6. The two sub-ports are released when data transfer completes.

B.2 Protocol Stacks

The base layer transports messages between the client and the server. Protocols below the base layer handle idiosyncrasies in the transport layer. For example, a TCP connection transports a byte stream; thus, the messages are prefixed with their length before they are sent over a TCP link.

B.2.1 Base Layer Packet Format

The base layer packet contains a prefix of four words, which are followed by a variable number of payload bytes. The prefix is used by the base layer; the payload is delivered to higher layers of the protocol stacks.

The prefix contains a string of ASCII characters comprising four blank-delimited words and a single trailing blank. (This allows for simple parsing in REXX.)

The four words of the prefix contain:

1. Destination sub-port.
2. Origin sub-port.
3. Packet sequence number. All packets sent to sub-port 0 are sent with sequence number 0, because they are independent transmissions.
4. Packet type.

By virtue of being character strings, all four fields are in network byte order.

Example:

```
0 0 0 cmd cp q t
```

B.2.2 Overview of Packet Types

These values are defined for the packet type field. The list is sorted alphabetically.

The word “file” should be taken in the most general meaning. It represents a collection of related data. The data might be the contents of a file, the lines that make up the response to a command, the drawing primitives to create a window, or a similar related set of data.

ack — Acknowledge receipt of a data packet.

cmd — Request the partner to issue a command and send the response back.

cncl — Terminate data transfer before the source signals end-of-file.

data — Send a data packet that is not the last transmission for a file.

datl — Send the last data packet of a file. This packet can contain no data if the file is empty or if it was not possible to determine that the previous data block was the last of the file at the time it was transmitted.

dats — Send file status information. This packet is sent before the data blocks of a file.

end — Notify the server of impending termination. This packet is sent by the client before it terminates.

eof	Notify the client that no data will be transmitted for a file. (cf. `datl`.)
err	Notify the partner of an error condition.
getf	Request the partner to retrieve the contents of a file.
getp	Request the partner to send the data produced by running a pipeline.
msg	Send a message for the programmer responsible for the partner.
pipe	Request the partner to set up a pipeline, read data into it from the partner, and return the output to the partner.
putf	Request the partner to store the contents of a file.
putp	Request the partner to pass data to a pipeline.

Most of the packets are sent to the control port to start a request, for example to initialize a pipeline segment. Only packets `ack`, `cncl`, `data`, `datl`, `dats`, `eof`, and `err` are sent to a sub-port other than the control port.

B.3 Transport Layer Protocols

The Plumbers' Workbench assumes that the transport layer is reliable and that it can buffer a few hundred bytes. A reliable transport layer delivers the bytes that are sent in the order they are sent, without corrupting them.

B.3.1 Transmission Control Program Transport Layer

Messages are prefixed by a four-byte binary length field, which is in network byte order. The length count includes the length of the four-byte length field.

On CMS, the length field is added and stripped by *CMS Pipelines*'s *tcpclient* device driver when it is invoked with an argument string that includes the keyword SF4.

Appendix C. Packet Formats

This appendix describes the contents of the payload field in the packets transmitted between the Plumbers' Workbench client and its server.

C.1 File Status Information

The file status is an ASCII character string that contains information about a file. It contains nine or more blank-delimited words:

1. Time of last access for the file (the `atime` in POSIX parlance). If the time of last access is not available, the time of last modification is sent.
2. Time of last modification for the file (the `mtime` in POSIX parlance).
3. Time of last control change for the file (the `ctime` in POSIX parlance). If the time of last control change is not available, the time of last modification is sent.
4. Time of file creation. If the time of creation is not available, a zero is sent.
5. The format of the file:

binary	Binary byte stream. The file has no record boundaries.
fixed	Fixed-format file. All records are the same length.
pctext	PC-style text file. The file contains lines, which are terminated by carriage return and line feed characters (X'0D0A').
text	Text file. The file contains lines, which are terminated by line feed characters (X'0A').
variable	Variable-format file. Records are of any length up to the logical record length for the file. Record boundaries are not preserved during transmission. Thus, there is no difference between formats "b" and "v" once the file is transmitted, but the "v" highlights the loss of information.

   The record format can be followed by a period and a code page number without intervening blanks.
6. Logical record length, if it is known; otherwise zero.
7. Number of records or lines, if it is known; otherwise zero.
8. Estimate of file size in bytes. The estimate can be high, but it should never be lower than the actual file size. A zero means that the file size is not known or cannot be estimated.
9. The POSIX-style path to the file. The path extends to the end of the packet or to the first null character. Note that POSIX file names can contain blanks.

C.2 Detailed Packet Formats

Each packet type is described in a separate section below. The description includes the purpose of each packet, its payload data (if any), and the expected response (if any).

C.2.1 Iam

Purpose: To notify the partner of the program's level.

Payload: The payload contains five blank-delimited words:

1. The constant "PWB"
2. The level of protocol implemented. The initial implementation sends "1.0000".
3. The date of the last significant change.
4. The system on which the program is running (the first word of the REXX source string).
5. The level of access authorization granted to the partner. In order of priority from most restrictive:

n	No access privileges are granted.
r	Files can be read.
w	Files can be written and erased.
c	Commands may be issued.
a	Any function can be requested.

Expected Response: An `Iam` packet from the partner.

Remarks: The partners send this packet concurrently. They serialize by receiving it. Either partner can terminate the connection if it decides that it is unable to generate packets in the format required by the partner, as defined by the protocol level it supports.

C.2.2 ack

Purpose: To acknowledge the receipt of a `data`, `datl`, `dats`, or `putf` packet. The sequence field sent is the sequence field of the last packet acknowledged.

Payload: This packet carries no payload data.

Expected Response: A packet carrying the next block of data (`data` or `datl`).

C.2.3 cncl

Purpose: To terminate data transfer prematurely. The `cncl` packet is sent instead of an `ack` packet when no further data is required for the transmission.

Payload: The payload data is in ASCII. The payload can contain an optional message in ASCII.

Expected Response: None.

C.2.4 cmd

Purpose: To issue a command and pass the response back as a file.

Payload: The payload data is in ASCII. The first word specifies the file format to be used in transmitting the response. The remainder of the payload specifies the command to be issued.

Expected Response: A dats packet.

C.2.5 data

Purpose: To transmit a block of data bytes to the partner.

Payload: The payload contains the data bytes as a byte stream. The payload can be blocked into fixed-size blocks or it can be unblocked. The data format is as specified in the dats packet at the beginning of the transmission.

Expected Response: An ack packet.

C.2.6 datl

Purpose: To transmit the last block of data bytes for a transmission to the partner.

Payload: The payload contains the last packet of the data in the byte stream. The payload can be empty.

Expected Response: An ack packet.

C.2.7 dats

Purpose: To transmit file status information. When the partner is requesting data to be transmitted, this packet indicates that the data transfer request can proceed (For example, that a file exists and that it can be transferred).

The dats packet precedes data packets for a transmission that is requested by the partner.

Payload: The payload contains the file status details in ASCII. The information is sent as blank-delimited words in the format defined in C.1, “File Status Information” on page 83.

Expected Response: An ack packet.

C.2.8 end

Purpose: To alert the partner that the program is terminating. This packet should only be sent between the control sub-ports.

Payload: The payload can contain an optional message in ASCII.

Expected Response: None.

C.2.9 eof

Purpose: To alert the partner that no data will be produced.

Payload: The payload can contain an optional message in ASCII.

Expected Response: None.

C.2.10 err

Purpose: To return an error condition to the partner. An err packet is sent in response to a request that cannot be processed or cannot be completed. For example, if a file does not exist, an err packet is returned instead of the dats packet, which would be sent to indicate that the request can be processed.

Payload: The payload is ASCII. The first word contains an error code:

again — The request cannot be processed at this time because of a resource shortage. It may be possible to process the request at a later time.

fileform — The format of the file name is not correct.

invreq — The request type is not recognized.

noauth — The partner was not authorized for the function requested.

nocmd — The command returned an error.

nofile — The file does not exist (or the program is not authorized to access it).

noopen — The file could not be opened.

nopipe — The pipeline could not be issued.

noread — The file could not be read.

Expected Response: None.

C.2.11 getf

Purpose: To request the transmission of the contents of a file.

Payload: The payload data is in ASCII. The first word specifies the required file format. The remainder of the payload specifies the file name in the standard format.

Expected Response: A dats packet.

C.2.12 getp

Purpose: To request the transmission of the data produced by a pipeline.

Payload: The payload data is in ASCII. The payload contains the format of the data being transferred followed by the pipeline to be issued. The format of the pipeline depends on the operating system running at the partner.

Expected Response: A dats packet.

C.2.13 msg

Purpose: Issue a message to the user of the client or to the programmer who is responsible for the server.

Payload: Message text in ASCII.

Expected Response: None.

C.2.14 pipe

Purpose: To request that a pipeline be set up to process data sent from the partner and return the output from this pipeline to the partner. This is accomplished by two sub-session tasks, the primary one to send data to the partner and the secondary one to accept the response from the partner. The origin port represents the primary task; it is the source of the data that should be written into the remote pipeline. The payload data contains the remote port number for the port to receive the output from the pipeline.

Payload: The payload data is in ASCII. The payload contains the format of the data being transferred, the number of a remote port, and the pipeline to be issued. The format of the pipeline depends on the operating system running at the partner.

Expected Response: An ack packet on the primary port. A dats packet on the secondary port.

C.2.15 putf

Purpose: To store the contents of a file.

Payload: The payload data is in ASCII. The information is sent as blank-delimited words in the format defined in C.1, “File Status Information” on page 83.

Expected Response: An ack packet.

C.2.16 putp

Purpose: To request that data transmitted be processed by a pipeline.

Payload: The payload data is in ASCII. The payload contains the format of the data being transferred followed by the pipeline to be issued. The format of the pipeline depends on the operating system running at the partner.

Expected Response: An ack packet.

Appendix D. Program Directory

The Plumbers' Workbench diskette distribution is described in this appendix.

D.1 Deliverables

The files listed below make up the deliverables for the Plumbers' Workbench. There is a section for each sub-directory.

D.1.1 Client

VMPIPE.EXE	The OS/2 command to run a pipeline segment on the server. This file may reside in any directory.
PWBCTL.CMD	The client version of the REXX program that controls the connection to the partner. At the time of publishing, this file must be located in the c:\jph\src\pwb\ directory.
SG244523 BOO	Online documentation. This file can be viewed on OS/2. It has been "stamped" for use with the viewer that is supplied with the collection kit CD-ROMs. Upload this file to a file type BOOK if you wish to use the host BookManager reader.

D.1.2 Server

The files that make up the server are stored in EBCDIC. They must be uploaded in binary. The file type is truncated to three characters in the OS/2 file; be sure to upload to the full file type.

PWB EXEC	Main server EXEC.
PWBCTL REXX	The server version of the REXX program that controls the connection to the partner.
PWBUME TEXT	The message repository.
PWBUME REPOS	Source file for the message repository.
PWBMETER REXX	Measure pipeline throughput.
PWBTCPVA REXX	Handle connection requests from TCP/IP. Create pipeline thread to service request.
PWBTCPAU REXX	Establish authorities for partner.
XMITMSG REXX	Issue messages from a repository with substitution.
PWBMESSA REXX	Issue an error message.

List of Abbreviations

IP	Internet protocol
ITSO	International Technical Support Organization
PC	Personal Computer (IBM)
PC-DOS	Personal Computer disk operating system
PTF	program temporary fix
PWB	Plumbers' WorkBench
R/W	read/write
REXX	restructured extended executor language
TCP	transmission control protocol (USA, DoD)
TCP/IP	Transmission Control Protocol/Internet Protocol
UNIX	an operating system developed at Bell Laboratories; a pun on MULTICS
VM	virtual machine (IBM System 370 & 390)
VMTOOLS	VM programs conference facility

Index

Special Characters

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

ITSO Technical Bulletin Evaluation RED000

International Technical Support Organization
Plumbers' Workbench
CMS Pipelines and OS/2 Pipe Synergy
December 1995

Publication No. SG24-4523-00

Your feedback is very important to help us maintain the quality of ITSO Bulletins. **Please fill out this questionnaire and return it using one of the following methods:**

- Mail it to the address on the back (postage paid in U.S. only)
- Give it to an IBM marketing representative for mailing
- Fax it to: Your International Access Code + 1 914 432 8246
- Send a note to REDBOOK@VNET.IBM.COM

Please rate on a scale of 1 to 5 the subjects below.
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction	____		
Organization of the book	____	Grammar/punctuation/spelling	____
Accuracy of the information	____	Ease of reading and understanding	____
Relevance of the information	____	Ease of finding information	____
Completeness of the information	____	Level of technical detail	____
Value of illustrations	____	Print quality	____

Please answer the following questions:

a) If you are an employee of IBM or its subsidiaries:

Do you provide billable services for 20% or more of your time? Yes____ No____

Are you in a Services Organization? Yes____ No____

b) Are you working in the USA? Yes____ No____

c) Was the Bulletin published in time for your needs? Yes____ No____

d) Did this Bulletin meet your needs? Yes____ No____

If no, please explain:

_____

_____

What other topics would you like to see in this Bulletin?

_____

_____

What other Technical Bulletins would you like to see published?

_____

Comments/Suggestions: (THANK YOU FOR YOUR FEEDBACK!)

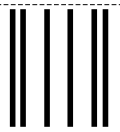
_____ _____
Name Address

_____ _____
Company or Organization

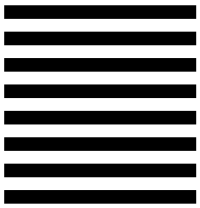
_____ _____
Phone No.

Cut or Fold
Along Line

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



NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES



BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM International Technical Support Organization
Mail Station P099
522 SOUTH ROAD
POUGHKEEPSIE NY
USA 12601-5400

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

Cut or Fold
Along Line

Printed in U.S.A.