Software Installer
Version 1.2 for OS/2

July 23, 1993

IBM Corporation
11000 Regency Parkway
Cary, North Carolina USA 27512-9968
Attention: Bill Orr, K43, Building 671-2Ba

This document is a PostScript file converted from a LIST3820 format of the EPFIPGR.INF online viewable *Software Installer Reference* for the installation and maintenance utility coded in Cary.

The format obtained in this LIST3820 is different from that obtained with IPFC. The format in this LIST3820 is not the designed format.

You should view the *Software Installer Reference* (EPFIPGR.INF) for the official document style and format.

We would appreciate any suggestions you can make for this information unit. Thanks for your help. Please return your comments to Bill Orr, K43/2Ba/671, 11000 Regency Parkway, Cary, N.C. 27511-9968. (BILL_ORR@VNET.IBM.COM)

Contents

Chapter 1. Trademarks

The following terms, denoted by an asterisk (*) on their first occurrences in this document, are trademarks or service marks of the IBM Corporation in the United States or other countries:

Common User Access
CUA
IBM
IBM PC Support/400
Operating System/2
Operating System/400
OS/2
Presentation Manager
SAA
Virtual Storage Extended
VSE/ESA
Workplace Shell

Chapter 2. Document Introduction

''''

Autolink to Sibling Window

RES='&hres.' VPX='25%' VPY='bottom' VPCX='75%'

- **How this document is organized** [**Link to heading 2.1 on page 2**]
- **Conventions used in this document** [**Link to heading 2.2 on page 3**]
- **Related information** [**Link to heading 2.3 on page 4**]

2.1 How This Document Is Organized

You should start with "Using Software Installer."

This document is divided into the following sections.

Trademarks [**Link to heading Chapter 1 on page 1**]
Contains a list of trademarks used in this document.

Document Introduction [**Link to heading Chapter 2 on page 2**]
Describes the organization and conventions of the document and lists related information.

Using Software Installer [**Link to heading Chapter 3 on page 5**]
Provides a brief description of what Software Installer is and a list of tasks you should complete to use Software Installer in your product.

Using the Diskette Generator to Create Diskettes or Upload to Hosts [**Link to heading Chapter 4 on page 14**]
Describes how to use the diskette-generating utility.

Accessing Maintenance Functions After Installation [**Link to heading 6.3 on page 28**]
Describes how to enable your product for delete, restore, and update services.

Catalog File [**Link to heading Chapter 5 on page 17**]
Describes how to create and use the catalog file.

Package File [**Link to heading Chapter 6 on page 26**]
Describes how to create and use the package file.

Description File [**Link to heading Chapter 7 on page 70**]
Describes how to create and use the description file.

Choosing How Your Product Installs [**Link to heading Chapter 8 on page 71**]
Describes how to tailor your product's installation.

Registering Installation Classes and Creating Objects for the Workplace Shell * [**Link to heading Chapter 9 on page 79**]
Describes how to enable your installation files to the OS/2* Workplace Shell.

Compressing Your Source Files [**Link to heading Chapter 10 on page 80**]
Tells how to pack the product files before shipping your product.

Updating Files That are in Use [**Link to heading Chapter 11 on page 82**]
Explains how Software Installer allows file replacement when some files are in use.

Using Hooks to Change Installation Directories [**Link to heading Chapter 12 on page 83**]
Explains how you can customize your installation for special directory processing.

Exiting During Installation Processing [**Link to heading Chapter 13 on page 84**]
Tells how to use installation exits to customize the installation of your product.

Installation Variables [**Link to heading Chapter 17 on page 118**]
Lists the installation variables that are available for use in installation exits.

Default Installation Exits [**Link to heading 13.9 on page 89**]
Lists the default installation exits that are available when you use Software Installer.

Servicing Your Shipped Product [**Link to heading Chapter 14 on page 101**]
Describes how you can use Software Installer to install updates of your shipped product.

Renaming Software Installer Files [**Link to heading Chapter 15 on page 105**]
Describes how to use the rename utility to avoid conflicts with other products that use Software Installer files.

Enabling Command Line Installation of Your Product [**Link to heading Chapter 16 on page 107**]
Provides information about how to enable installation from the command line.

Glossary [**Link to heading Chapter 19 on page 122**]
Defines terms used in this document.

2.2 Conventions Used in This Document

This online document uses hypertext links to access additional information on a subject you want to know more about. If a phrase in a panel is highlighted (if it appears in a different color from surrounding text), that means there is more information available. Select the phrase, and the additional information is displayed.

For example, if you want to get started building the installation files for your product, you should follow the **Step-by-Step Instructions for Using Software Installer**[**Link to heading 3.7 on page 10**]. Select the highlighted hypertext phrase to go to the list of instructions. When you want to return, press the Escape key. If you select several hypertext links, one after another, you must press the Escape key several times to get back to where you started.

You can select highlighted phrases with either the keyboard or a mouse. To use the keyboard, press the Tab key to move from one hypertext phrase to another in a panel, then press Enter to select a particular phrase. If you have a mouse, double-click on a highlighted phrase to select it.

In the Catalog File and Package File sections, a panel displays the syntax of each entry type. All keywords are listed for each of these entry types along with either an optional or required designation. The *(optional)* or *(required)* is not part of the entry.

This document uses example source code to demonstrate how to use installation services. Code examples shown in this document appear in a font that is different from the rest of the text in the document. For example:

```
EXIT = 'EXEC fg fs %EPFICURPWS%'
```

More information about the use of online documentation is available in the *IPF Reference* that is one of the OS/2 Toolkit Information units.

2.3 Related Information

Information in the following documents might be useful when you are building and customizing an application installation.

Refer to the following documents for information about the Common User Access* (CUA*) user interface.

- *SAA* CUA Advanced Interface Design Reference* (SC34-4290)
- *SAA CUA Guide to User Interface Design* (SC34-4289)

Refer to the following documents for information about OS/2*.

- *Using OS/2 2.0* (G362-0007)
- *IBM Operating System/2* Local Area Network Server Version 2.0 LAN Requester* (S04G-1037)
- *IBM Operating System/2 Version 2.0 Application Design Guide* (S10G-6260)
- *IBM Operating System/2 Version 2.0 Command Reference Manual* (S10G-6313)
- *IBM Operating System/2 Version 2.0 Control Program Programming Reference* (S10G-6263)
- *IBM Operating System/2 Version 2.0 Information Presentation Facility Guide and Reference* (S10G-6262)
- *IBM Operating System/2 Version 2.0 PM* Programming Reference Volume 1* (S10G-6264)
- *IBM Operating System/2 Version 2.0 PM Programming Reference Volume 2* (S10G-6265)
- *IBM Operating System/2 Version 2.0 PM Programming Reference Volume 3* (S10G-6272)
- *IBM Operating System/2 Version 2.0 Programming Guide Volume 1* (S10G-6261)
- *IBM Operating System/2 Version 2.0 Programming Guide Volume 2* (S10G-6494)
- *IBM Operating System/2 Version 2.0 Programming Guide Volume 3* (S10G-6495)
- *IBM Operating System/2 Version 2.0 Procedures Language 2, REXX Reference* (S10G-6268)
- *IBM Operating System/2 Version 2.0 Procedures Language 2, REXX User's Guide* (S10G-6269)
- *IBM Operating System/2 Version 2.0 System Object Model Guide and Reference* (S10G-6309)

Chapter 3. Using Software Installer

,,,,,,,

Autolink to Sibling Window

RES='3100' VPX='25%' VPY='bottom' VPCX='75%' VPCY='100%

3.1 Overview of Software Installer

This section describes the features and supported environments of Software Installer.

Note: For a description of *how* you use Software Installer, see the **Step-by-Step Instructions for Using Software Installer**[**Link to heading 3.7 on page 10**].

Software Installer Version 1.2 for OS/2* (Software Installer) is a programming tool that helps you automate the installation process for applications you develop. You can use Software Installer to provide your users with an easy and consistent way to install and maintain workstation applications you develop for the OS/2 environment.

Software Installer provides services for installing, updating, restoring, and deleting the workstation parts of products that you distribute on diskette or initially install from a host system.

With Software Installer you can install software on an individual workstation, a LAN connected to the workstation, or a combination of both.

You can create a standard installation procedure that runs on multiple workstations, without requiring input from your users.

Supported Drive Environments:

Software Installer supports installation from the following drive environments:

- CD-ROM
- Diskette
- Local Area Network (LAN)

Supported Host Environments:

Software Installer supports installation from the following host environments:

- Multiple Virtual Storage (MVS)
- Operating System/400* with IBM PC Support/400*
- Virtual Machine (VM)
- Virtual Storage Extended (VSE/ESA)* Version 1.3.0

Major functions:

- **CID**[**Link to heading Chapter 20 on page 123**] enabled
- Provides multi-threaded execution. You can install one product while using another.
- Compresses your product files using Software Installer's utility or your own.
- Decompresses compressed files during installation using Software Installer's utility or your own.
- Maintenance functions include:
  - Updates an installed product for corrective service or the next release of a product
  - Restores a previous release of a product
  - Deletes a product from the user's workstation
- Runs exits supplied by Software Installer during installation that can:
  - Set environment variables
  - Modify the user's CONFIG.SYS file
  - Add and delete information from the OS2.INI file or an application's own INI file
  - Create, register, deregister, and delete Workplace Shell* objects
- Runs user-supplied exits.
- Provides non-interactive installation of a product.
- Allows the user to selectively install components of a multiple-component product.
- Allows up to 20 directories for installation destinations.
- Provides for tailoring of the initial installation screen to:
  - Display up to 100 bit maps
  - Animate the bit maps
  - Display an optional information window that is useful for displaying a README file
- Replaces product files that are in use.
- Installs from diskette without installing Software Installer.
- Provides a utility for renaming Software Installer files you redistribute.

Operating Environment

Machine Requirements

- A personal computer running OS/2 Version 2.0

  Note: For more information on the OS/2 Version 2.0 machine requirements, refer to the *IBM OS/2 Version 2.0 Information and Planning Guide* (G326-0160).

- 2 MB of hard disk space above that required by OS/2
- .5 MB of memory above that required by OS/2
- 2-button mouse or other pointing device

Program Requirements

- IBM Operating System/2* (OS/2) Version 2.0 operating system. ServicePak XR06055 is recommended.

  Note: Without the ServicePak, your installation will not return the appropriate results from user exits even though the exit completes correctly. Also, Workplace Shell objects created during an installation will not use the specified icon.

- IBM Developer's Toolkit for OS/2 Version 2.0.

  Note: The Developer's Toolkit for OS/2 is required for creating a customized product installation. However, end users do not need to have the toolkit on their systems to install your products.

- To install from an AS/400 host system, you need IBM PC Support/400* (Program Number 5738-PC1) Version 2 Release 2.
- To install from a VM, MVS, or VSE host system, you need IBM Extended Services for OS/2 2.0 with Communications Manager installed.

3.2 Components of Software Installer and What They Do

The following lists divide Software Installer files into four major groups. Understanding the purpose of these files will help you understand how to use Software Installer.

Note: For a description of *what to do* to use Software Installer, see the **Step-by-Step Instructions for Using Software Installer**[**Link to heading 3.7 on page 10**].

Base [**Link to heading 3.3 on page 7**]
The as-shipped bit maps and executables that provide the installation and maintenance processing.

Customizing [**Link to heading 3.4 on page 8**]
The files that are needed to rename and customize Software Installer files for your particular application.

Samples [**Link to heading 3.5 on page 9**]
Working sample catalog, package, and description files for a fictional software product. You can use these as a model for your files.

Documentation [**Link to heading 3.6 on page 10**]
The documentation that is part of Software Installer.

3.3 Base Components of Software Installer

You can use Software Installer ″out of the box″ with these files.

INSTALL.EXE
The program that the user runs to start the installation from a drive. It does the following:

- Finds an available hard disk drive with enough space to temporarily hold the installation utility files
- Unpacks the installation utility files onto the user′s hard disk
- Starts EPFIDLDS
- Deletes the files from the temporary directory after EPFINSTS has finished

EPFIDLDS.EXE
The program that provides the initial installation onto the workstation by starting EPFINSTS. EPFIDLDS contains bit maps that you can modify by changing parameters in the IIRC.RC resource file. BLDRC.CMD is provided for compiling the modified resource file.

EPFINSTS.EXE
The program that provides the processing which is referred to in this document as the installation and maintenance utility. This program reads the catalog, package, and description files you create and performs the installation.

DISKGEN.EXE
The program that you run to generate diskettes on a workstation or upload files to a host system. You can start this program from the Diskette Generator object in the Software Installer folder or from the command line.

EPFIEXTS.DLL
Dynamic link library that contains the entry points to the default installation exits.

EPFIHPLB.HLP
The online help for the installation and maintenance utility. To see an explanation of the end-user error and information messages:

1. Select **General Help...** from the Help menu on the Installation and Maintenance window
2. Select **Contents** from the Options menu
3. Expand the Error messages entry at the bottom of the Contents list

4. Select the help for the message you want to see

EPFIMSG.MSG
The installation and maintenance utility message file.

EPFIPII.DLL
Dynamic link library that contains program integrated information.

EPFIPRCS.EXE
Executable that is started by EPFINSTS.EXE.

EPFIPACK.EXE
The program that compresses source files.

EPFISRSP.DAT
Response file that contains information to automate the installation of Software Installer.

EPFIUPCK.EXE
The program that decompresses source files packed using EPFIPACK.EXE.

3.4 Customizing Components of Software Installer

Use the following parts of Software Installer to tailor installation and maintenance services for your product.

ISREN.CMD
A utility for **renaming** [**Link to heading Chapter 15 on page 105**] Software Installer files for shipment with your products. You should use the directory that the files are renamed into for your working directory.

BLDINST.CMD
Command file that creates a new INSTALL.IN_ (packed file you distribute containing installation and maintenance files) from your renamed copy of installation and maintenance utility files.

BLDRC.CMD
Command file that rebinds the customized IIRC.RC to the renamed copy of EPFIDLDS.EXE.

EPFIHOOK.C
C program for the **hook** [**Link to heading Chapter 24 on page 127**] that is called each time the directories in the Install - directories window are changed.

EPFIHOOK.DEF
Module definition for the directories hook program.

EPFIHOOK.MAK
Make file for the directories hook program.

EPFIICIS.ICO
The install icon registered to the Workplace Shell.

EPFIICO1.ICO
The install folder icon.

EPFIIIRS.RES
A compiled installation resource file.

EPFISINC.PKG
A package file that enables the installation and maintenance utility to **install and register Software Installer on your customer's workstation**[**Link to heading 6.3 on page 28**] when your product is installed.

EPFRCOPY.EXE
The rename utility's executable file.

IIRC.RC
The resource file you customize to tailor the renamed EPFIDLDS.EXE bit maps and default startup options.

IIRCARW.BMP
An arrow bit map.

IIRCEXCL.BMP
An exclamation mark bit map.

IIRCH.H
A header file that defines entry points for IIRC.RC.

IIRCINST.BMP
″Installation″ bit map.

VARS.OBJ
Object file containing **PUTVAR, GETVAR, and GETOWNER functions**[**Link to heading 13.5 on page 86**]. You must use a 32-bit compiler, such as the IBM C-Set/2 compiler, when compiling your exits that use these functions.

VARSFP.H
C language header file for the PUTVAR, GETVAR, and GETOWNER functions.

3.5 Sample Files

The following sample files are in the \SAMPLES subdirectory of the directory you installed Software Installer into. Double click on the **Samples** icon in the Software Installer folder to run the sample installation.

INVACDLL.DL_
Packed inventory DLL
INVDSC.DSC
Description file for the inventory part of SSS
NFOODDOC.DO_
Packed non-food component DOC files
NFOODEXE.EX_
Packed non-food component EXE files
NPRSHDOC.DO_
Packed non-perishable component DOC files
NPRSHEXE.EX_
Packed non-perishable component EXE files
PERSHPAK.PA_
Packed perishable component PAK file list
PRICEDLL.DL_
Packed pricing DLL files
PRICEDOC.DO_
Packed pricing DOC files
PRICEDSC.DSC
Description file for the pricing part of SSS
PRICEEXE.EX_
Packed pricing EXE files
PRICEPKG.PKG
Package file for the pricing part of SSS
PROMODOC.DO_
Packed promotiional DOC files

PROMOEXE.EX_
Packed promotiional EXE files

SUPERMKT.ICF
Catalog file for the SuperMarket Software System (SSS)

3.6 Documentation files

EPFIHLIB.HLP
The online help.

EPFIHLIB.ITL
The source for the online help. Your can change this to customize the help for your product's installation. See the READ.ME file for instructions for recompiling the online help.

EPFIPGR.INF
The online reference.

READ.ME
A text file with an overview of Software Installer functions and instructions for getting started using the product

3.7 Step-by-Step Instructions for Using Software Installer

To enable a product for installation by the installation and maintenance utility, you should complete the following tasks after you have installed Software Installer on your workstation.

1. Run the ISREN.CMD command file to **rename**[**Link to heading Chapter 15 on page 105**] the Software Installer files that will be distributed with your product. You can start the rename utility by double clicking on the **Rename Software Installer** icon in the Software Installer folder. Running the rename utility gives you a copy of the necessary Software Installer files in a working directory.

   If you do not specify a directory for the renamed files, one will be created for you. Do not specify the directory you are renaming from as the directory you are renaming to.

   The remaining tasks should be done in the directory containing the renamed installation and maintenance utility files.

2. Create **catalog**[**Link to heading 5.2 on page 17**], **package**[**Link to heading 6.2 on page 27**], and **description files**[**Link to heading 7.2 on page 70**].

3. **Test**[**Link to heading 3.8 on page 13**] the catalog, package, and description files by running the renamed EPFINSTS.EXE.

4. If you want to change the processing for the Install - directories dialog, **rebuild the renamed EPFIHOOK.DLL**[**Link to heading Chapter 12 on page 83**].

5. Modify the **IIRC.RC**[**Link to heading Chapter 8 on page 71**] resource file. You must create any **response file** [**Link to heading 16.4 on page 113**] **STARTPARM_RESPFILE**[**Link to heading 8.8 on page 77**] and information file(s) **INFOn_FILE** [**Link to heading 8.6 on page 75**] referenced in IIRC.RC.

6. Compile the IIRC.RC resource file by using the BLDRC.CMD command file.

7. Test the customized IIRC.RC using the renamed EPFIDLDS.EXE. Check the location and contents of the bit maps and information windows you are using in your product's installation.

8. If you are shipping your product for use in different languages, you must customize a IIRC.RC, catalog, and package file for each language This will result in a unique INSTALL.IN_ file for each language.

The IIRC.RC should contain the translated text for the initial Install window title, the translated Information window text, the translated product name, and any bitmap customizations for the language.

The catalog file should contain the translated NAME keyword values and DESCRIPTION keyword values.

The package file should contain the correct volume labels for the language as well as translated values for keywords.

9. If you are using Software Installer to install your product from a diskette, continue with the following steps. Otherwise, continue with the ″Host-specific installation steps.″

Diskette-specific installation steps

1. Run BLDINST.CMD to create an INSTALL.IN_ file that contains the packed installation and maintenance utility files.
2. Prepare one diskette for each **DISK entry** [**Link to heading 6.15 on page 46**] in the package file by formatting and labelling the diskettes so that the label of each diskette matches the NAME keyword of a DISK entry in the package file. You can set the volume label when the diskette is formatted or use the OS/2 **LABEL**[**Link to footnote, ID=os2com on page 13**] command.
3. If your product is on more than one CD-ROM or diskette, ensure that the VOLUME label of each diskette matches the VOLUME keyword of the corresponding DISK entry.

   If you are distributing your product on one CD-ROM or diskette, you do not need VOLUME keywords for either the DISK or FILE entries in your package file.
4. Update each **FILE entry**[**Link to heading 6.25 on page 48**] in the package file to contain a VOLUME keyword that indicates which diskette the file is to be copied to.
5. Copy the following files onto the distribution diskette in the following order.
   - The INSTALL.EXE
   - The INSTALL.IN_
   - The catalog, package, and description files
6. If you are using the **Diskette Generator to create diskettes**[**Link to heading Chapter 4 on page 14**]: start DISKGEN.EXE from the command line or double click on the Diskette Generator object in the Software Installer folder.

   If you specified the /U option, you are prompted for a new path and name for the output package file, you have two choices:
   - Entering A:,input_package_file_name.pkg> will place the output file onto the distribution diskette with the same name as the input package file.
   - Pressing Enter without a path and name results in the output file being placed in the current directory with a file name the same as the input file and an extension of .GEN. You must then copy the .GEN file to the distribution diskette with a .PKG extension.
7. If you are *not* using the Diskette Generator to create diskettes:
   a. **Compress**[**Link to heading 10.1 on page 80**] your product source files.
   b. Copy the packed product source files onto the distribution diskette,

   Note: The INSTALL.EXE, INSTALL.IN_, catalog, package, and description files must be on the first diskette.
8. Test your diskette installation files by typing A:\ and INSTALL.EXE at the OS/2 command prompt.

Host-specific installation steps

If you are using Software Installer to install your product from a host system, continue with the following steps.

1. Upload the following files to the host systems that your product can be installed from.

   Note: When uploading workstation files to the host in ASCII format, you should specify the CRLF option with the SEND command.

   - The renamed EPFIDLDS.EXE. This file can be renamed to anything you want the user to type when installing your product. Upload this file in binary format.
   - Any **response file**[**Link to heading 16.4 on page 113**] or **information files**[**Link to heading 8.6 on page 75**] that you created to customize your installation in step 5 above. Upload these files in ASCII format.

     Note: These files must be on the end-user's workstation when the renamed EPFIDLDS.EXE is started. You might want to provide a CMD file for the end user to download the EPFIDLDS and the other files.
   - The package, catalog, and description files. Upload these files in ASCII format.
   - The following renamed installation and maintenance utility files in binary format.

     You must upload these renamed utility files with qualifiers specific to the host system that indicate whether the file is translatable. The following table shows the source file qualifiers as they are shipped in the IIRC.RC file. You can **specify your own qualifiers** [**Link to heading 8.2 on page 72**] by modifying a string table in your renamed IIRC.RC.

Base parts (non-translatable)	NLS parts (translatable)
---	---
EPFIEXTS.DLL	EPFIHPLB.HLP
EPFINSTS.EXE	EPFIMSG.MSG
EPFIPRCS.EXE	EPFIPII.DLL
EPFIUPCK.EXE	

Use the following qualifiers depending on the host system you are uploading to.

	Base part qualifiers	NLS part qualifiers
----	---	---
MVS	SEPFBASE	SEPFBENU
VM	BASEPF11	ENUEPF11
VSE	BASEPF11	ENUEPF11

2. If you are using the **Diskette Generator to upload your product files to a host system**[**Link to heading Chapter 4 on page 14**]:
   a. Start DISKGEN.EXE from the command line or double click on the Diskette Generator object in the Software Installer folder.
   b. Upload the output package file generated by the Diskette Generator to the host system in ASCII. Use the same name for the uploaded file as your input package file.
3. If you are *not* using the diskette-generator utility to upload your installation to host: upload the compressed product source files in binary format.

4. Test your host installation files by downloading the renamed EPFIDLDS.EXE file, response file, and other information files, Start the renamed EPFIDLDS at the OS/2 command prompt.

Table 1. Footnote Panels for Current Heading

Footnote ID	Footnote
os2com	Refer to the *IBM Operating System/2 Version 2.0 Commands Reference* for more information.

3.8 Testing Your Catalog, Package, and Description Files

Starting the renamed EPFINSTS displays the Installation and Maintenance window. If you have previously installed a product, the product name is listed in this window. You should complete the following steps to test your catalog, package, and description files.

1. Select the **Open catalog** choice from the File pull-down menu.
2. Select the **Drive** choice from the cascaded menu.
3. Enter the drive that contains your catalog file in the **Drive** field.
4. Enter the file name of the catalog in the **Filename** field.
5. Select **Open**.

   The Installation and Maintenance window should be displayed with the name of your product listed.
6. Select **Install** from the Action pull-down menu.
7. Check the Product number, Version, and Feature displayed in the Install window.
8. Select **OK** to install your product.
9. When the Install - directories window is displayed, you have successfully tested the syntax of your catalog, package, and description files. Select **Cancel** from the Install - directories window and **Exit** from the File pull-down menu on the Installation and Maintenance window. You should proceed with the remaining steps in the ″Step-by-Step Instructions.″

Chapter 4. Using the Diskette Generator to Create Diskettes or Upload to Hosts

,,,,,,,

Autolink to Sibling Window

RES='&hres.' VPX='25%' VPY='bottom' VPCX='75%' VPCY='100%

4.1 Description of the Diskette Generator

The Diskette Generator is a Software Installer supplied utility that can be started from the OS/2 command line or from the *Diskette Generator* object in the Software Installer folder on the desktop.

It performs the following functions:

- Automatically generates diskette(s) for OS/2 applications.
- Outputs a package file with the path and name you specify.

  This package file's VOLUME keyword of the FILE entry identifies the diskettes that the product files are on. The DATE and TIME keywords in the FILE entry are updated to reflect the creation dates and times of the product files.
- Uploads OS/2 files to a VM or MVS host for tape distribution.
- Compresses your files using Software Installer's compression program or any other compression program with a fully qualified name or located in the current directory

4.2 Syntax and Parameter Definitions

Note: If you start the Diskette Generator from the desktop object, you will be prompted for the required parameters.

Use the following syntax to start DISKGEN.EXE from the command line.

DISKGEN /<options> <system> <input package file> <destination> <source_directory>

Notice the following requirements for running DISKGEN.EXE.

- You must use five parameters. You can use a period (.) as a token for the source_directory.
- There must be one space between DISKGEN and the /<options>.
- The parameters must be in the order shown in the syntax.
- The parameters must be separated with a single space.
- Parameters are not case-sensitive.

If you specify your own packing utility when prompted, you must edit the output package file. Change the value of the UNPACK keyword of the FILE entry to the name of your unpacking utility.

The five required parameters are:

<options> Required
Characters that tell the installation and maintenance utility how to build your diskettes. A slash (/) must precede the characters. More than one option can be specified. There should be no spaces in the <options> parameter.

Valid values are:

A
Specifies that the diskette generator should use the PWS keyword of the FILE entry in the input package file to further qualify the source_directory parameter. This means that the directory in the PWS keyword is appended to the source_directory for the search for your product files.

If you specified the PWSPATH keyword in a FILE entry which has a PWS keyword, the file specified in the PWS must be fully qualified.

For example, if you used DISKGEN /UA DRIVE MUPKG.PKG A: E:\MYPROD with the following package file segment, the diskette generator would search E:\MYPROD\MYFILE.EXE. The search would fail because your files are in the directory you specified in the PATH entry with the FILE keyword.

```
PATH
   FILE = 'E:/MYPROD/PRODDIR'
FILE
   WHEN = 'OUTOFDATE',
   REPLACEINUSE = 'I U D R',
   UNPACK = 'YES',
   SOURCE = 'DRIVE:MYFILE.EX_',
   PWSPATH = FILE,
   VOLUME = 'MBBENU01',
   PWS = 'MYFILE.EXE',
   DATE = '921010',
   TIME = '0000',
   SIZE = '1'
```

In cases where the destination is a <drive:>, you should fully qualify the PWS keyword (in the above example PWS = 'PRODDIR/MYFILE.EXE') because the diskette generator does not use the FILE specification of the PATH entry.

D
Specifies to delete the files named in the input package file after the diskette is built or the files are uploaded to a host system.

U
Specifies to update the DATE keyword in the output package file to the dates of the files packed by the diskette generator. Also specifies to update the SIZE keyword to the bytes required by the product files before compression. If you do not specify the U option, the DATE and SIZE keywords in the output package file are the dates and sizes specified in the DATE and SIZE keywords of the input package file.

If you specify the /U option, you are prompted for a new path and name for the output package file, you have two choices:

- Entering A:\<input_package_file_name.pkg> will place the output file onto the distribution diskette with the same name as the input package file.
- Pressing Enter without a path and name results in the output file being placed in the current directory with a file name the same as the input file and an extension of .GEN. You must then copy the .GEN file to the distribution diskette with a .PKG extension.

<system> Required
The system environment of the source files.

Valid values are:

DRIVE
MVS
VM

<input package file> Required
The workstation name of your product's **package file**[**Link to heading Chapter 6 on page 26**].

<destination> Required
The target host session or drive location that you want the files uploaded to or diskette generated on.

Valid formats for the different target systems are:

<drive:>
drive

For example:

A:

<session_id:project.group>
MVS

For example:

B:project1.group2

<session_id:filemode>
VM

For example:

B:A

<source_directory>
The workstation directory containing the files specified in the input package file. You can use a period (.) for a token if you do not specify a directory.

Chapter 5. Catalog File

,,,,,,,

Autolink to Sibling Window

RES='4100' VPX='25%' VPY='bottom' VPCX='75%' VPCY='100%

- **Overview** [**Link to heading 5.1 on page 17**]
- **Creating a catalog file** [**Link to heading 5.2 on page 17**]
- **Example** [**Link to heading 5.3 on page 18**]

Entry types

- **CATALOG** [**Link to heading 5.4 on page 20**]
- **INCLUDE** [**Link to heading 5.9 on page 21**]
- **PACKAGE** [**Link to heading 5.14 on page 22**]

5.1 Overview of the Catalog File

The catalog file contains product information that the installation and maintenance utility needs to install and maintain a product. The catalog file can contain this information about more than one product.

A catalog file consists of multiple occurrences of entry types. Each entry type has a different function. Occurrences of entry types are called entries. Each entry includes a series of keywords followed by an equals (=) sign and a keyword value.

Entries have the following form.

```
entry_type
  keyword1=value1,
  keyword2=value2,
  .
  .
  keywordn=valuen
```

There are three valid entry types for a catalog file:

- CATALOG
- INCLUDE
- PACKAGE

5.2 Creating a Catalog File

The first step in enabling your product for installation with the installation and maintenance utility is to create a catalog file in the directory that contains the **renamed**[**Link to heading Chapter 15 on page 105**] installation and maintenance utility files.

A catalog file can contain information about more than one product.

You store information such as the product name, product number, and feature number in the catalog file. The installation and maintenance utility uses the NAME, NUMBER, and FEATURE keywords of the PACKAGE entry to determine if a product is currently installed on the workstation.

In this file, you also provide the name of the package files and description files to be used when the products are installed.

You ship the catalog file in text format on the host tape or on disk along with the product files.

Use your editor to create a catalog file with the following characteristics.

- Contains a CATALOG entry type if you want to specify a name and description of your product.
- Contains an INCLUDE entry type if there are additional catalog files to be included for processing with this catalog file. Do not use more than five levels of INCLUDEs.
- Contains a PACKAGE entry type for each product. There can be a maximum of 79 PACKAGE entries in each catalog file.
- The PACKAGEFILE, PKGDESCRFILE, and HOSTLOC keywords in the **PACKAGE entry** [**Link to heading 5.14 on page 22**] identify the source location of the package file, description file, and product files.
- Is a sequential text file, if the installation is from a host system.
- Is in fixed or variable format if the installation is from a host system.
- Has a maximum record length of 255 characters if installation is from a host system.
- Has a file extension of ICF if the installation is from a diskette.

Additional considerations

- You can have no more than one entry per line, but each entry can occupy one or more lines.
- Keywords can appear in any order.
- You can use blank spaces between the entry type and the keyword, around the equals sign, and after the commas. You can use blank lines between entries and between keywords.
- You must enclose keyword values in single quotes if they contain commas, blanks, or single quotes. When dealing with values that contain single quotes, use two single quotes to represent a quote mark in the value.
- Keyword values can span lines, in which case the value must be enclosed in a single quote. Trailing and leading blanks on each line are included in the value.
- A line with an asterisk as the first non-blank character indicates a comment. All text on the line is ignored.
- You can create one catalog file that supports installation from both a host system and a **drive**[**Link to heading Chapter 21 on page 124**].

5.3 Example of a Catalog File

```
*********************************************************************
*                                                                   *
*      SuperMarket Systems Software                                 *
*                                                                   *
*      Sample Software Installer catalog file                       *
*                                                                   *
*              5621-434 (C) COPYRIGHT IBM CORP 1993                 *
*      ALL RIGHTS RESERVED. LICENSED MATERIALS - PROPERTY OF IBM     *
*                                                                   *
*********************************************************************

* Define the catalog.
* There is only one CATALOG entry in each catalog file.

CATALOG
   * Define a user-friendly name for the catalog.
   NAME         = 'SuperMarket Systems Software',
   * Define a description of the catalog
   DESCRIPTION  = 'This is the catalog of a fictional product called SuperMarket Systems Software.
   The catalog file represents a typical catalog file used by Software Installer to install the
   product''s files. See the Software Installer license agreement for an explanation of your rights
   governing uses of sample files.'

* Define the package.
* There can be one or more PACKAGE entries in each catalog file.
* In this sample there is one for the Pricing System package and
* one for the Inventory Control System.

PACKAGE
   * Define a name for the package.  This is the
   * name that appears in the catalog list.
   NAME         = 'Pricing System',
   * Define the product number of the files in the package.
   NUMBER       = '1234-567',
   * Define the version(01), release(01), and modification
   * number(00) of the files in the package.
   VRM          = '010200',
   * Define the feature number of the files in the package.
   FEATURE      = '0001',
   * Define the source location of the files.
   HOSTLOC      = 'MVS: SMS.V1R1M0.; VM: ; VSE: SUPERMKT.V1R1M0.; DRIVE: ;',
   * Define the name of the package file.
   PACKAGEFILE  = 'MVS: FILES(PRICEPKG);
                   VM: PRICEPKG PACKAGE *;
                   VSE: PRICEPKG.PACKAGE;
                   DRIVE: PRICEPKG.PKG',
   * Define the name of the description file.
   PKGDESCRFILE = 'MVS: FILES(PRICEDSC);
                   VM: PRICEDSC DESCRPTN *;
                   VSE: PRICEDSC.DESCRPTN;
                   DRIVE: PRICEDSC.DSC',
   * Define the amount of disk space required to install
   * this package on the user's workstation.
   SIZE         = '500000'

PACKAGE
   NAME         = 'Inventory Control System',
   NUMBER       = '1234-567',
   VRM          = '010200',
```

```
FEATURE      = '0002',
HOSTLOC      = 'MVS: SMS.V1R1M0.; VM: ; VSE: SUPERMKT.V1R1M0.; DRIVE: ;',
PACKAGEFILE  = 'MVS: FILES(INVPKG);
                VM: INVPKG PACKAGE *;
                VSE: INVPKG.PACKAGE;
                DRIVE: INVPKG.PKG',
PKGDESCRFILE = 'MVS: FILES(INVDSC);
                VM: INVDSC DESCRPTN *;
                VSE: INVDSC.DESCRPTN;
                DRIVE: INVDSC.DSC',
SIZE         = '5000000'
```

* End of SUPERMKT.ICF catalog file

5.4 CATALOG Entry Type

,,,,,,,

Autolink to Sibling Window

```
RES='&hres.' VPX='25%' VPY='bottom' VPCX='75%' VPCY='100%
```

- **Syntax** [**Link to heading 5.5 on page 20**]
- **Description** [**Link to heading 5.6 on page 20**]
- **Keywords** [**Link to heading 5.7 on page 20**]
- **Example** [**Link to heading 5.8 on page 21**]

5.5 CATALOG Entry Type Syntax

```
CATALOG
  DESCRIPTION ='<description of the catalog contents>', (optional)
  NAME        ='<name of the catalog>' (optional)
```

5.6 CATALOG Entry Type Description

The CATALOG entry type provides information about what the catalog contains. This entry type can be used to specify a name and a 512-character description of the contents of the catalog.

5.7 CATALOG Entry Type Keywords

DESCRIPTION — optional
A description of the contents of the catalog file. The value of this keyword is a text string with a maximum length of 512 characters. This text is displayed when the user selects the **Description** push button on the Open drive catalog window or the Open host catalog window. If this keyword is omitted, a "Description is not available" message is displayed when the user selects the **Description** push button.

NAME — optional
A descriptive name for the catalog file. The value of this keyword is a text string with a maximum of 60 characters. This keyword value is displayed by the installation and maintenance utility as the catalog title when the catalog file is opened. If this keyword is omitted, a generic title of ″Product catalog″ is used.

5.8 CATALOG Entry Type Example

```
* There is only one CATALOG entry in each catalog file.

CATALOG
   * Define a user-friendly name for the catalog
   NAME        = 'SuperMarket Systems Software',
   * Define a description of the catalog
   DESCRIPTION = 'This catalog represents a fictional product called
                  SuperMarket Systems Software.  It is intended to
                  represent a typical catalog file used by Software Installer
                  to install software files.  See the Software Installer license
                  agreement for an explanation of your rights governing
                  sample files.'
```

5.9 INCLUDE Entry Type

,,,,,,,

Autolink to Sibling Window

```
RES='&hres.' VPX='25%' VPY='bottom' VPCX='75%' VPCY='100%
```

- **Syntax** [**Link to heading 5.10 on page 21**]
- **Description** [**Link to heading 5.11 on page 21**]
- **Keyword** [**Link to heading 5.12 on page 22**]
- **Example** [**Link to heading 5.13 on page 22**]

5.10 INCLUDE Entry Type Syntax

```
INCLUDE
  NAME=='<file name of included file>' (required)
```

5.11 INCLUDE Entry Type Description

The INCLUDE entry type specifies any additional catalog file to be included when the installation and maintenance utility processes the original catalog file.

You should use one INCLUDE entry for each catalog file you want to be processed with the original catalog file.

Note: You should not have more than five levels of included catalog files.

The following represents the maximum permitted five levels of catalog inclusion:

```
LEVEL1.ICF includes LEVEL2.ICF
  LEVEL2.ICF includes LEVEL3.ICF
    LEVEL3.ICF includes LEVEL4.ICF
      LEVEL4.ICF includes LEVEL5.ICF
```

However, you can include several catalog files from one level.

5.12 INCLUDE Entry Type Keyword

NAME — required
Name of a catalog file to be included. You must specify the full name of the catalog file to be included.

This format allows you to create one package file that works for all source systems.

You should use the following format for this keyword.

NAME = 'MVS: mvsname; VM: vmname; VSE: vsename; DRIVE: drivename'

If the catalog is being transferred from an MVS host, *MVS: mvsname* is used; if from a VM host, *VM: vmname* is used; if from a VSE host, *VSE: vsename* is used; if from a drive, *DRIVE: drivename* is used.

You should use a single forward slash for the drive directory separator. The installation and maintenance utility will convert this forward slash to a backslash.

Do not specify a system if there is no catalog file to be included from that system.

The maximum length is 255 characters including the single quotes.

5.13 INCLUDE Entry Type Example

```
* Define the catalog file to include with the current catalog file.
* All locations must be fully qualified.
INCLUDE
 NAME = 'MVS: PROJ.GROUP.TYPE(INCLCAT); VM: INCLCAT CATALOG *; VSE: LIB.SUBLIB.INCLCAT.CATALOG ;
   DRIVE: /INCLCAT.ICF'
```

5.14 PACKAGE Entry Type

,,,,,,,

Autolink to Sibling Window

```
RES='&hres.' VPX='25%' VPY='bottom' VPCX='75%' VPCY='100%
```


5.15 PACKAGE Entry Type Syntax

```
PACKAGE
  FEATURE     ='<feature number>', (required)
  NAME        ='<product name>', (required)
  NUMBER      ='<program number>', (required)
  PACKAGEFILE ='<file name of package file>', (required)
  SIZE        ='<product size in bytes>', (required)
  VRM         ='<version, release, and modification numbers>', (required)
  HOSTLOC     ='<location of product files>', (conditionally required)
  PKGDESCRFILE='<file name of description file>', (optional)
  UPDATE      ='<yes or no>' (optional)
```

5.16 PACKAGE Entry Type Description

The PACKAGE entry type provides the installation and maintenance utility with information such as the product name, product number, feature number, and version number, as well as the name of the package file and description file.

You must create a PACKAGE entry for each product that the installation and maintenance utility is to install.

5.17 PACKAGE Entry Type Keywords

FEATURE — required
Number used to distinguish between different features of a product. For example, your product can have different feature numbers for its language availability (C, REXX, or COBOL), distribution media (host tape or diskette), and density of the media. The feature number is a text string with a maximum length of 20 characters.

NAME — required
Name of the product. The name is a text string with a maximum length of 60 characters. This name is displayed in the product list of the Installation and Maintenance window.

Not all product names fit on all windows because the installation and maintenance utility windows use the system proportional font. Although there is sufficient space for most product names, product owners should be sure that the product name they choose will fit on all the windows.

NUMBER — required
Product number. The number is a text string with a maximum of 20 characters.

PACKAGEFILE — required
Name of the package file. It is a text string with a maximum length of 255 characters. The keyword specification is dependent on the source system of the install. The format of this keyword allows one catalog file to be used regardless of the source system. Use the following format:

PACKAGEFILE='<MVS: mvsname; VM: vmname; VSE: vsename; DRIVE: drivename>'

You should use a single forward slash for the drive directory separator. The installation and maintenance utility will convert this forward slash to a backslash.

Do not specify a system if there is no package file on that system.

See the HOSTLOC keyword form more information on qualifying this value.

SIZE — required
Disk space in bytes required to install the entire product. This value must be a string of not more than nine digits. Commas and other separators are not allowed: *2000000* is valid, but *2,000,000* is not. This

keyword is displayed on the Disk space window and is used to determine whether the package can be installed on a specified hard drive.

VRM — required
Version, release, and modification number. The format of this keyword is VVRRMM, where V, R, and M are digits only. For instance

VRM=010100

is a valid VRM.

HOSTLOC — conditionally required
Source location of the specified files. The maximum length is 255 characters. The value specified in this keyword is concatenated to the beginning of all the file names specified by the following keywords:

- PACKAGEFILE and PKGDESCRFILE keywords of the **PACKAGE entry type**[**Link to heading 6.40 on page 58**] in the catalog file
- SOURCE keyword of the **FILE entry type**[**Link to heading 6.25 on page 48**] in the package file
- NAME keyword of the **INCLUDE entry type**[**Link to heading 6.30 on page 55**] in the package file
- SOURCE keyword of the **PACKFILE entry type**[**Link to heading 6.40 on page 58**] in the package file

The HOSTLOC keyword contains the high-level file name qualifiers that do not change for each file. This keyword allows administrators to install the product from any MVS data set, VSE library, or drive without having to change the file names in every FILE, INCLUDE, PACKFILE, and PACKAGE entry.

If you specify a HOSTLOC for the MVS and VSE systems, do not use a fully qualified file name. If you do not specify a HOSTLOC for the MVS and VSE systems, you must fully qualify the file location with the file name. Do not specify a HOSTLOC keyword for files on VM.

PKGDESCRFILE — optional
Name of the **description**[**Link to heading Chapter 7 on page 70**] file. It is dependent on the source system of the install. The format of this keyword allows one catalog file to be used regardless of the source system. Do not specify a system if there is no description file on that system. The maximum length is 255 characters.

The contents of the description file appears in the Product description window after the user selects **Product description** from the Details pull-down menu. If you omit the PKGDESCRFILE keyword and the user selects **Product description**, the installation and maintenance utility indicates that a description is not available.

See the HOSTLOC keyword form more information on qualifying this value.

UPDATE — optional
Specifies that the package can only be installed as an update. Set this value to YES when you are shipping a correction that must be applied to a previously installed version. Valid values for this keyword are:

YES The product can be updated only if a previous version is already installed. The Install action will not be allowed. If this keyword is set to YES and the product is not previously installed, the product name does not appear in the product listing in the Installation and Maintenance window.

NO (default) The product can be installed or updated. No check for a previous version is performed.

5.18 PACKAGE Entry Type Example

```
* There can be one or more PACKAGE
* entries in each catalog file.

PACKAGE
   * Define a name for the package.  This is the
   * name that appears in the catalog list.
   NAME          = 'Pricing System',
   * Define the product number of the files in the package.
   NUMBER        = '1234-567',
   * Define the version, release, and modification numbers of
   * the files in the package.
   VRM           = '010200',
   * Define the feature number of the files in the package.
   FEATURE       = '0001',
   * Define the source location of the files.
   HOSTLOC       = 'MVS: SMS.V1R1M0.; VM: ; VSE: ; DRIVE: ;',
   * Define the name of the package file.
   PACKAGEFILE   = 'MVS: FILES(PRICEPKG);
                    VM: PRICEPKG PACKAGE *;
                    VSE: SUPERMKT.V1R1M0.PRICEPKG.PACKAGE;
                    DRIVE: PRICEPKG.PKG',
   * Define the name of the description file.
   PKGDESCRFILE = 'MVS: FILES(PRICEDSC);
                    VM: PRICEDSC DESCRPTN *;
                    VSE: SUPERMKT.V1R1M0.INVDSC.DESCRPTN;
                    DRIVE: PRICEDSC.DSC',
   * Define the amount of disk space required to install
   * this package.
   SIZE          = '500000'
```

Chapter 6. Package File

,,,,,,,

Autolink to Sibling Window

RES='5100' VPX='25%' VPY='bottom' VPCX='75%' VPCY='100%

6.1 Overview of the Package File

This file describes the files that make up your product. The package file provides Software Installer with all of the information needed to install, update, restore, or delete a product on the workstation. The installation and maintenance utility does the following:

1. Parses for syntactic correctness.

   Errors are displayed in a secondary window on the Install-File Errors window. See **Testing Your Catalog, Package, and Description Files[Link to heading 3.8 on page 13]**.

2. Builds a list of entries for processing.

   INCLUDE entries are included in the processing list as they are encountered. OPTIONS and PATH entry keyword values are stored in memory for use during processing.

3. Sequentially processes the entry list that was built. This list does not include any INCLUDE, OPTIONS, or PATH entries.

You ship the package file on the product tape or diskettes with your source files. When you release product corrections, you ship a new package file that describes how the corrections are to be applied (transferred to the workstation).

On tape-shipped products, the package file is unloaded into the same target libraries as the product files. On diskette-shipped products, the package file is shipped on the first product diskette.

A package file consists of multiple occurrences of entry types. Each entry type has a different function. Occurrences of entry types are called entries. Each entry includes a series of keywords followed by an equal (=) sign and a keyword value.

Entries have the following form.

```
entry_type
  keyword1=value1,
  keyword2=value2,
  .
  .
  keywordn=valuen
```

There are eleven package file entry types.

- ADDCONFIG
- COMPONENT
- DISK
- EXIT
- FILE
- INCLUDE
- OPTIONS
- PACKFILE
- PATH
- SERVICELEVEL
- UPDATECONFIG

6.2 Creating a Package File

When you are first starting, create a simple package file and add to it as you go along. A basic package file requires only a FILE entry for each file of your product.

The installation and maintenance utility validates the syntax of the package file and keyword values at the time it is read. If errors are found, the installation and maintenance utility displays them in a window. Installation cannot begin until you correct the errors in the package file.

After testing your installation files, you can begin using the predefined **installation exits** [**Link to heading 13.9 on page 89**] to automatically update the workstation CONFIG.SYS file, the OS2.INI configuration file, or the Workplace Shell. You might also decide to divide the files of your product into separately installable components by adding some COMPONENT entries. As a final step, you might create some product-specific installation exits to completely automate the installation.

Use your editor to create a package file with the following characteristics:

- Sequential text file if installation is from a host system
- Fixed or variable format if installation is from a host system

- Record length of up to 255 characters if installation is from a host system
- Has a file extension of PKG if installation is from a diskette

Additional considerations

- You can have no more than one entry per line, but each entry can occupy one or more lines.
- Keywords can appear in any order.
- You can use blank spaces between the entry type and the keyword, around the equals sign, and after the commas. You can use blank lines between entries and between keywords.
- You must enclose values in single quotes if they contain commas, blanks, or single quotes. When dealing with values that contain single quotes, use two single quotes to represent a quote.
- Values can span lines, in which case the value must be enclosed in a single quote. Trailing and leading blanks on each line are included in the value.
- A line with an asterisk as the first non-blank character indicates a comment. All text on the line is ignored.

6.3 Accessing Maintenance Functions After Installation

If you want your customers to be able to use the delete, update, and restore Software Installer services, you must do either of the following:

- Supply a customized xxxIDLDS.EXE with the STARTPARM_ACTION set to the desired action (U=Update, R=Restore, D=Delete).
- Allow your users access to the Installation window, which is supplied by the renamed EPFINSTS.EXE. This window has a menu bar that contains the maintenance functions.

If your product is installed from diskette and you want to allow your users access to your renamed EPFINSTS.EXE, so that they can use the maintenance functions, the installation and maintenance utility files must be installed on the user's workstation. You can enable this by including EPFISINC.PKG in your package file. When the installation and maintenance utility processes EPFISINC.PKG in your product's package file, it will install and register Software Installer on your customer's workstation.

If you do not include EPFISINC.PKG, installation and maintenance files are automatically deleted from the user's workstation by INSTALL.EXE.

If your product is installed from a host system, there is a possibility of the user entering a path on the Install - directories window that is different than that you have in the package file. This could result in two sets of installation and maintenance files on the user's workstation. You may want to write an exit to search for the files and delete them.

The EPFISINC.PKG file is renamed when you run the ISREN.CMD. You must make changes to the renamed EPFISINC.PKG file before you include it in your package file. The required changes are described in comments in the renamed EPFISINC.PKG file. Doing a global change of EPF to your prefix is not recommended because there are %EPFxxxxxx% installation variables that must not be changed.

After making the specified changes, include the renamed package file in your product's package file. For example, if your product prefix used in the rename process is ABC and your product is installable from a VM host and a drive, you should use the following INCLUDE entry in your package file.

```
INCLUDE
  NAME='VM:ABCISINC PACKAGE;DRIVE:ABCISINC.PKG'
```

6.4 Example of a Package File for a Product without Components

```
*********************************************************************
*                                                                   *
*      SuperMarket Systems Software                                 *
*                                                                   *
*      Sample Software Installer package file                       *
*                                                                   *
*      This package file illustrates a product without components.  *
*                                                                   *
*             5621-434   (C) COPYRIGHT IBM CORP 1991, 1993.         *
*      ALL RIGHTS RESERVED. LICENSED MATERIALS - PROPERTY OF IBM     *
*                                                                   *
*********************************************************************

* Define the service level.
SERVICELEVEL
   * Set the service level to a value that indicates
   * this is the first release.
   LEVEL = '0000000'

* Define which diskette contains the files to be installed.
DISK
   * Define a user-friendly name for the diskette.
   * Software Installer prompts the user for this name.
   NAME = 'Software Installer Disk 1',
   * The files are on a diskette whose label is INSTALL1.
   VOLUME = 'INSTALL1'

* Define the default directories and labels to appear in the
* Install - directories window.  Software Installer prompts the user for
* four directories.
PATH
   FILE = 'C:/SUPERMKT/PRICE/EXE',
   FILELABEL = 'Executable directory:',
   WORK = 'C:/SUPERMKT/PRICE/DLL',
   WORKLABEL = 'DLL directory:',
   AUX1 = 'C:/SUPERMKT/PRICE/DOC',
   AUX1LABEL = 'Documentation directory:',
   AUX2 = 'C:/SUPERMKT/PRICE/DATA',
   AUX2LABEL = 'Data directory:'

* Define a FILE entry for each file to be transferred.
* In this sample, EXE, DLL, and DOC files are transferred.
FILE
   * Define a descriptive name for the file.
   * This keyword is only used by the installation variables
   * that are set during installation.
   NAME = 'PRICEEXE',
   * Define the type of transfer as binary.
   DOWNLOAD = 'STATIC',
   * Install this file when the file with this name
   * on the workstation is old
   * or does not exist on the workstation.
   WHEN = 'OUTOFDATE',
   * Set the date of the file to January 28, 1992.
   DATE = '920128',
   * Set the time of the file to 08:30am.
   TIME = '0830',
   * Do not set the file's hidden bit.
   HIDDEN = 'NO',
```

```
    * Do not set the file's read-only bit.
    READONLY = 'NO',
    * Install this file in the FILE directory.
    PWSPATH = 'FILE',
    * Because the file was packed using EPFIPACK,
    * unpack it using EPFIUPCK.
    UNPACK = 'YES',
    * Define the possible locations of the packed file.
    SOURCE = 'MVS:FILES(PRICEEXE);
              VM:PRICEEXE EXEBIN *;
              VSE:PRICEEXE.EXEBIN;
              DRIVE:PRICEEXE.EX_',
    * Define which diskette the file exists on.
    VOLUME = 'INSTALL1',
    * Define the name given to this file on the workstation.
    PWS = 'PRICE.EXE',
    * Run the exit when this file is installed.
    EXITWHEN = 'INSTALL',
    * Ignore all errors from the exit.
    EXITIGNOREERR = 'YES',
    * Add the FILE directory to the PATH installation variable.
    EXIT = 'UPDATECONFIG SET PATH=%EPFIFILEDIR%;'

FILE
    DOWNLOAD = 'STATIC',
    WHEN = 'OUTOFDATE',
    DATE = '920128',
    TIME = '0830',
    HIDDEN = 'NO',
    READONLY = 'NO',
    PWSPATH = 'WORK',
    UNPACK = 'YES',
    SOURCE = 'MVS:FILES(PRICEDLL);
              VM:PRICEDLL DLLBIN *;
              VSE:PRICEDLL.DLLBIN;
              DRIVE:PRICEDLL.DL_',
    VOLUME = 'INSTALL1',
    PWS = 'PRICE.DLL',
    EXITWHEN = 'INSTALL',
    EXITIGNOREERR = 'YES',
    EXIT = 'UPDATECONFIG LIBPATH=%EPFIWORKDIR%;'

FILE
    NAME = 'PRICEDOC',
    DOWNLOAD = 'STATIC',
    WHEN = 'OUTOFDATE',
    DATE = '920128',
    TIME = '0830',
    HIDDEN = 'NO',
    READONLY = 'NO',
    PWSPATH = 'AUX1',
    UNPACK = 'YES',
    SOURCE = 'MVS:FILES(PRICEDOC);
              VM:PRICEDOC DOCBIN *;
              VSE:PRICEDOC.DOCBIN;
              DRIVE:PRICEDOC.DO_',
    VOLUME = 'INSTALL1',
    PWS = 'PRICE.DOC'
```

```
* In the following FILE entry, no files are transferred.
* Only an exit is run.
* The exit registers 'Promotional Inventory Control' in
* the 'SuperMarket Systems' folder.
* The folder uses the system provided folder icon.
FILE
   EXITWHEN='INSTALL',
   EXITIGNOREERR='NO',
   * This exit registers the program 'Pricing System'
   * to the 'SuperMarket Systems' folder.
   EXIT='ADDPROG.PM "SuperMarket Systems" "Pricing System" %EPFIFILEDIR%\PRICE.EXE'

* Delete 'Pricing System' from the 'SuperMarket Systems'
* folder.  The folder is deleted when the last item
* in the folder is deleted.
FILE
   EXITWHEN='DELETE',
   EXITIGNOREERR='YES',
   EXIT='DELETEPROG "SuperMarket Systems" "Pricing System"'

* End of sample package file for product with no component.
```

6.5 Example of a Package File for a Product with Components

```
*********************************************************************
*                                                                   *
*      SuperMarket Systems Software                                 *
*                                                                   *
*      Sample Software Installer package file                       *
*                                                                   *
*      This package file illustrates a product with components.     *
*                                                                   *
*            5621-434   (C) COPYRIGHT IBM CORP 1991, 1993.          *
*      ALL RIGHTS RESERVED. LICENSED MATERIALS - PROPERTY OF IBM     *
*                                                                   *
*********************************************************************

* Define the service level.
SERVICELEVEL
   * Set the service level to a value that indicates
   * this is the first release.
   LEVEL = '0000000'

* Define which diskette contains the files to be installed.
DISK
   * Define a user-friendly name for the diskette.
   * Software Installer prompts the user for this name.
   NAME = 'Software Installer Disk 1',
   * The files are on a diskette whose label is INSTALL1.
   VOLUME = 'INSTALL1'

* Define the default directories and labels to appear in the
* Install - directories window.
* Software Installer prompts the user for seven directories.
PATH
   FILE = 'C:\SUPERMKT\INVNTRY\EXE',
   FILELABEL = 'Executable directory:',
   WORK = 'C:\SUPERMKT\INVNTRY\DLL',
   WORKLABEL = 'DLL directory:',
   AUX1 = 'C:\SUPERMKT\INVNTRY\DOC',
   AUX1LABEL = 'Documentation directory:',
   AUX2 = 'C:\SUPERMKT\INVNTRY\DATA\PROMO',
   AUX2LABEL = 'Promotional data: ',
   AUX3 = 'C:\SUPERMKT\INVNTRY\DATA\PERISH',
   AUX3LABEL = 'Perishables data: ',
   AUX4 = 'C:\SUPERMKT\INVNTRY\DATA\NOPERISH',
   AUX4LABEL = 'Non-perishables data: ',
   AUX5 = 'C:\SUPERMKT\INVNTRY\DATA\NONFOOD',
   AUX5LABEL = 'Non-food data: '


************************************************************
*                  COMPONENT     ONE                       *
* ---------------------------------------------------------- *
*                    Inventory Accounting                   *
*                                                           *
* This component is required by the other components        *
* and is installed when one of the others is installed.     *
************************************************************
*
* Define the 'Inventory Accounting' component.
* All files after this COMPONENT entry but before the next
* COMPONENT entry are part of 'Inventory Accounting'.
```

```
COMPONENT
   * Define the name of this component as you want it to
   * appear in the components list.
   NAME = 'Inventory Accounting',
   * Define an ID for this component.
   ID = 'INVACCT',
   * Do not display this component in the components list.
   DISPLAY = 'NO',
   * Define the amount of space on the workstation required
   * to install this component.
   * This component does not have a DESCRIPTION because
   * it is not displayed.
   SIZE = '500000'

* Define a FILE entry for each file of this component
* to be transferred.
FILE
   * Install this file when the file on the workstation is old
   * or the file does not exist on the workstation.
   WHEN = 'OUTOFDATE',
   * Set the date of the file to January 28, 1992.
   DATE = '920128',
   * Set the time of the file to 08:30am.
   TIME = '0830',
   * Install this file in the WORK directory.
   PWSPATH = 'WORK',
   * Because the file was packed using EPFIPACK,
   * unpack it using EPFIUPCK.
   UNPACK = 'YES',
   * Define the possible locations of the packed file.
   SOURCE = 'MVS:FILES(INVACDLL);
             VM:INVACDLL DLLBIN *;
             VSE:INVACDLL.DLLBIN;
             DRIVE:INVACDLL.DL_',
   * Define which diskette the file exists on.
   VOLUME = 'INSTALL1',
   * Define the name given to this file on the workstation.
   PWS = 'INVENTAC.DLL',
   * Run the exit when this file is installed.
   EXITWHEN = 'INSTALL',
   * Ignore all errors from the exit.
   EXITIGNOREERR = 'YES',
   * Add the WORK directory to the LIBPATH installation variable.
   EXIT = 'UPDATECONFIG LIBPATH=%EPFIWORKDIR%;'


************************************************************
*                  C O M P O N E N T     T W O             *
* ---------------------------------------------------------- *
*                Promotional Inventory Control               *
*                                                            *
************************************************************
COMPONENT
   NAME = 'Promotional Inventory Control',
   ID = 'PROMO',
   * Display this component in the components list.
   DISPLAY = 'YES',
   * When this component is installed, also install the
   * 'Inventory Accounting' component.
```

```
    REQUIRES = 'INVACCT',
    SIZE = '500000',
    * Define a description for this component.
    DESCRIPTION = 'Promotional Inventory Control manages the
    items in a store that are promotional or seasonal.'

* Define a FILE entry for the EXE and DOC files of the
* Promotional Inventory Control component.

FILE
    WHEN = 'OUTOFDATE',
    PWSPATH = 'FILE',
    UNPACK = 'YES',
    SOURCE = 'MVS:FILES(PROMOEXE);
              VM:PROMOEXE EXEBIN *;
              VSE:PROMOEXE.EXEBIN;
              DRIVE:PROMOEXE.EX_',
    VOLUME = 'INSTALL1',
    PWS = 'PROMOEXE',
    DATE = '920128',
    TIME = '0830',
    EXITWHEN = 'INSTALL',
    EXITIGNOREERR = 'YES',
    EXIT = 'UPDATECONFIG SET PATH=%EPFIFILEDIR%;'

FILE
    WHEN = 'OUTOFDATE',
    PWSPATH = 'AUX1',
    UNPACK = 'YES',
    SOURCE = 'MVS:FILES(PROMODOC);
              VM:PROMODOC DOCBIN *;
              VSE:PROMODOC.DOCBIN;
              DRIVE:PROMODOC.DO_',
    VOLUME = 'INSTALL1',
    PWS = 'PROMODOC',
    DATE = '920128',
    TIME = '0830'

* In the following FILE entry, no files are transferred.
* Only an exit is run.  The exit adds the installation variable
* SUPERMKT_PROMO_DATA to the user's CONFIG.SYS file with a
* value of the AUX2 directory.
FILE
    * Call the exit when the user is performing an
    * install or an update.
    EXITWHEN = 'INSTALL || UPDATE',
    EXITIGNOREERR = 'YES',
    * This exit defines an installation variable in
    * the user's CONFIG.SYS file.
    EXIT = 'ADDCONFIG SET SUPERMKT_PROMO_DATA=%EPFIAUX2DIR%'

* In the following FILE entry, no files are transferred.
* The ADDPROG exit registers 'Promotional Inventory Control' to
* the 'SuperMarket Systems' folder.  The folder uses the system
* provided folder icon.
FILE
    EXITWHEN = 'INSTALL',
    EXITIGNOREERR = 'NO',
    EXIT = 'ADDPROG PM "SuperMarket Systems" "Promotional Inventory Control"
```

```
    %EPFIFILEDIR%\PROMO.EXE {MB_CANCEL "Promotional Inventory Control" "Sample Program"'

* In the following FILE entry, no files are transferred.
* DELETEPROG deletes 'Promotional Inventory Control' from the
* 'SuperMarket Systems' folder.  The folder is deleted
* when the last item in the folder is deleted.
FILE
    EXITWHEN = 'DELETE',
    EXITIGNOREERR = 'YES',
    EXIT = 'DELETEPROG "SuperMarket Systems" "Promotional Inventory Control"'


************************************************************
*              C O M P O N E N T     T H R E E             *
* -------------------------------------------------------- *
*              Perishable Food Inventory Control           *
************************************************************
COMPONENT
    NAME = 'Perishable Food Inventory Control',
    ID = 'PERISH',
    DISPLAY = 'YES',
    REQUIRES = 'INVACCT',
    SIZE = '500000',
    DESCRIPTION = 'Perishable Food Inventory Control manages
    the items in a store that are perishable, such as fresh produce,
    meats, and dairy products.'

* The following is an example of multiple files in a
* single packed file.
* Two files, PERISH.EXE and PERISH.DOC are contained
* in the packed file.
* The PACKFILE keyword causes the file to be unpacked,
* and the individual file entries indicate that they
* are found in the packed file named in the
* PACKID keyword.  If the file was packed using your
* own utility, unpack it by using the name of your
* unpacking utility in the UNPACK keyword.  For example,
* UNPACK='PKUNZIP.EXE -o %EPFICURUPS% %EPFICURUPDIR%'
* would cause the PKUNZIP utility to be used
* to unpack the file specified by the EPFICURUPS
* installation variable into the directory specified
* by the EPFICURUPDIR installation variable.
* You should use the -o option for PKUNZIP to overwrite
* existing files.  If you use another unpacking utility,
* you should also use any option that specifies overwriting
* existing files.

PACKFILE
    SOURCE = 'MVS:FILES(PERSHPAK);
              VM:PERSHPAK PAKBIN *;
              VSE:PERSHPAK.PAKBIN;
              DRIVE:PERSHPAK.PA_',
    ID = 'PERSH'

* FILE entry for the EXE file of the Perishable Food Inventory Control
FILE
    WHEN = 'OUTOFDATE',
    PACKID = 'PERSH',
    PWSPATH = 'FILE',
```

```
    PWS = 'PERISH.EXE',
    DATE = '920128',
    TIME = '0830',
    EXITWHEN = 'INSTALL',
    EXITIGNOREERR = 'YES',
    EXIT = 'UPDATECONFIG SET PATH=%EPFIFILEDIR%;'

* FILE entry for the DOC file of the Perishable Food Inventory Control
FILE
    WHEN = 'OUTOFDATE',
    PACKID = 'PERSH',
    VOLUME = 'INSTALL1',
    PWSPATH = 'AUX1',
    PWS = 'PERISH.DOC',
    DATE = '920128',
    TIME = '0830'

FILE
    EXITWHEN = 'INSTALL || UPDATE',
    EXITIGNOREERR = 'YES',
    EXIT = 'ADDCONFIG SET SUPERMKT_PERISH_DATA=%EPFIAUX3DIR%'

FILE
    EXITWHEN = 'INSTALL || UPDATE',
    EXITIGNOREERR = 'NO',
    EXIT = 'ADDPROG PM "SuperMarket Systems" "Perishable Food Inventory Control"
    %EPFIFILEDIR%\PERISH.EXE (MB_CANCEL "Perishable Food Inventory Control" "Sample program"'

FILE
    EXITWHEN = 'DELETE',
    EXITIGNOREERR = 'YES',
    EXIT = 'DELETEPROG "SuperMarket Systems" "Perishable Food Inventory Control"'

************************************************************
*              C O M P O N E N T     F O U R               *
* ---------------------------------------------------------- *
*           Non-Perishable Food Inventory Control            *
************************************************************
COMPONENT
    NAME = 'Non-Perishable Food Inventory Control',
    ID = 'NPER',
    DISPLAY = 'YES',
    REQUIRES = 'INVACCT',
    SIZE = '500000',
    DESCRIPTION = 'Non-Perishable Food Inventory Control manages the items in a store
    that have a long shelf life, such as cereals, canned goods, and dried foods.'

FILE
    WHEN = 'OUTOFDATE',
    PWSPATH = 'FILE',
    UNPACK = 'YES',
    SOURCE = 'MVS:FILES(NPRSHEXE);
              VM:NPRSHEXE EXEBIN *;
              VSE:NPRSHEXE.EXEBIN;
              DRIVE:NPRSHEXE.EX_',
    VOLUME = 'INSTALL1',
    PWS = 'NPERISH.EXE',
    DATE = '920128',
    TIME = '0830',
```

```
    EXITWHEN = 'INSTALL',
    EXITIGNOREERR = 'YES',
    EXIT = 'UPDATECONFIG SET PATH=%EPHFILEDIR%;'

FILE
    WHEN = 'OUTOFDATE',
    PWSPATH = 'AUX1',
    UNPACK = 'YES',
    SOURCE = 'MVS:FILES(NPRSHDOC);
              VM:NPRSHDOC DOCBIN *;
              VSE:NPRSHDOC.DOCBIN;
              DRIVE:NPRSHDOC.DO_',
    VOLUME = 'INSTALL1',
    PWS = 'NPERISH.DOC',
    DATE = '920128',
    TIME = '0830'

FILE
    EXITWHEN = 'INSTALL || UPDATE',
    EXITIGNOREERR = 'YES',
    EXIT = 'ADDCONFIG SET SUPERMKT_NOPERISH_DATA=%EPHAUX4DIR%'

FILE
    EXITWHEN = 'INSTALL || UPDATE',
    EXITIGNOREERR = 'NO',
    EXIT = 'ADDPROG PM "SuperMarket Systems" "Non-Perishable Food Inventory Control"
    %EPHFILEDIR%\NPERISH.EXE (MB_CANCEL "Non-Food Inventory Control" "Sample program"'

FILE
    EXITWHEN = 'DELETE',
    EXITIGNOREERR = 'YES',
    EXIT = 'DELETEPROG "SuperMarket Systems" "Non-Perishable Food Inventory Control"'


************************************************************
*              C O M P O N E N T     F I V E               *
* -------------------------------------------------------- *
*               Non-Food Inventory Control                 *
************************************************************
COMPONENT
    NAME = 'Non-Food Inventory Control',
    ID = 'NONFOOD',
    DISPLAY = 'YES',
    REQUIRES = 'INVACCT',
    SIZE = '500000',
    DESCRIPTION = 'Non-Food Inventory Control manages the non-food items in a store,
    such as paper products, magazines, and pet supplies.'

FILE
    WHEN = 'OUTOFDATE',
    PWSPATH = 'FILE',
    UNPACK = 'YES',
    SOURCE = 'MVS:FILES(NFOODEXE);
              VM:NFOODEXE EXEBIN *;
              VSE:NFOODEXE.EXEBIN;
              DRIVE:NFOODEXE.EX_',
    VOLUME = 'INSTALL1',
    PWS = 'NONFOOD.EXE',
    DATE = '920128',
```

```
    TIME = '0830',
    EXITWHEN = 'INSTALL',
    EXITIGNOREERR = 'YES',
    EXIT = 'UPDATECONFIG SET PATH=%EPFIFILEDIR%;'

FILE
    WHEN = 'OUTOFDATE',
    PWSPATH = 'AUX1',
    UNPACK = 'YES',
    SOURCE = 'MVS:FILES(NFOODDOC);
              VM:NFOODDOC DOCBIN *;
              VSE:NFOODDOC.DOCBIN;
              DRIVE:NFOODDOC.DO_',
    VOLUME = 'INSTALL1',
    PWS = 'NONFOOD.DOC',
    DATE = '920128',
    TIME = '0830'

FILE
    EXITWHEN = 'INSTALL || UPDATE',
    EXITIGNOREERR = 'YES',
    EXIT = 'ADDCONFIG SET SUPERMKT_NONFOOD_DATA=%EPFIAUX5DIR%'

FILE
    EXITWHEN = 'INSTALL || UPDATE',
    EXITIGNOREERR = 'NO',
    EXIT = 'ADDPROG PM "SuperMarket Systems" "Non-Food Inventory Control"
    %EPFIFILEDIR%\NONFOOD.EXE (MB_CANCEL "Non-Food Inventory Control" "Sample program"'

FILE
    EXITWHEN = 'DELETE',
    EXITIGNOREERR = 'YES',
    EXIT = 'DELETEPROG "SuperMarket Systems" "Non-Food Inventory Control"'

* End of package file for product with five components
```

6.6 ADDCONFIG Entry Type

,,,,,,,

Autolink to Sibling Window

```
RES='&hres.' VPX='25%' VPY='bottom' VPCX='75%' VPCY='100%
```


- **Syntax** [**Link to heading 6.7 on page 39**]
- **Description** [**Link to heading 6.8 on page 40**]
- **Keywords** [**Link to heading 6.9 on page 41**]

6.7 ADDCONFIG Entry Type Syntax

```
ADDCONFIG
  VAR             ='<lValue>', (conditionally required)
  ADDSTR          ='<line to process>', (optional)
  ADDWHEN         ='<INSTALL|UPDATE|ALWAYS|NEVER|RESTORE|DELETE|<user_defined_variable>>', (optional)
  DELETEWHEN      ='<NEVER|ALWAYS|DIREMPTY|INSTALL|UPDATE|RESTORE|DELETE|<user_defined_variable>>', (optional)
  LOCATION        ='<END|BEGIN>', (optional)
  FILESEARCHPOS   ='<AFTER|BEFORE>', (optional)
  FILESEARCHSTR   ='<search string>', (conditionally required)
  FILESEARCHSTROCC='<LAST|FIRST>', (optional)
  REPLACEOCC      ='<LAST|FIRST|ALL>', (optional)
  TARGETDIR       ='<BOOT|<path>>', (optional)
  UNIQUE          ='<YES|NO>' (optional)
```

6.8 ADDCONFIG Entry Type Description

You can use the ADDCONFIG entry type to:

- Add a line to the beginning or end of the user's CONFIG.SYS file
- Add a new line before or after a target string that you specify
- Delete a line in the user's CONFIG.SYS file
- Replace the first, last, or all lines in the user's CONFIG.SYS file
- Specify which paths to change the CONFIG.SYS files in

The deletion feature uses the REM command to remove the specified line from the CONFIG.SYS. The REMed line is labelled with the product name being processed so that users can better assess the effect of a product's installation on their CONFIG.SYS. A line that has been removed is not used in subsequent searches unless the "REM" is also specified as part of the FILESEARCHSTR.

If your product's installation requires modification of the user's CONFIG.SYS file and the user does not select the Update CONFIG.SYS check box, an updated configuration file named CONFIG.ADD is copied from memory to the directory containing the CONFIG.SYS that needs to be modified.

If the user does select the Update CONFIG.SYS check box, the installation utility updates the user's configuration file and copies the old CONFIG.SYS to CONFIG.BAK in the directory where the CONFIG.SYS was modified.

The installation and maintenance utility uses the following precedence in positioning the line to add to CONFIG.SYS. (Default values are underlined.)

1. Searches for FILESEARCHSTR
   - Uses FILESEARCHSTROCC (LAST|FIRST) to find the FILESEARCHSTR
   - Uses FILESEARCHPOS (AFTER|BEFORE) to find the VAR relative to the FILESEARCHSTR
2. Uses LOCATION (END|BEGIN) if FILESEARCHSTR is not found or is not specified

6.9 ADDCONFIG Entry Type Keywords

You can use the following keywords with the ADDCONFIG entry type to modify the user's CONFIG.SYS file during installation and maintenance processing.

Keyword values are strings with a maximum length of 1024 characters.

VAR — conditionally required
VAR specifies the lValue (left side of the equals sign) which identifies the line you want added to or deleted from the CONFIG.SYS.

VAR is required if you do not specify REPLACEOCC.

For example:

```
ADDCONFIG
  VAR = 'FILES'
```

FILESEARCHSTR — conditionally required
When used with REPLACEOCC, FILESEARCHSTR specifies the line to replace. This keyword is required when you use the REPLACEOCC keyword. You must specify the entire line as the value of FILESEARCHSTR.

When you are adding a new line, FILESEARCHSTR can be any part of a line you want to find.

Use FILESEARCHSTROCC and FILESEARCHPOS to specify where, relative to the FILESEARCHSTR, to add the new line. The FILESEARCHSTROCC specifies whether to use the first or last occurrence of the FILESEARCHSTR. FILESEARCHPOS specifies whether to put the new line before or after that FILESEARCHSTR.

ADDSTR — optional
ADDSTR specifies the value of the VAR you are adding to the CONFIG.SYS.

The string consists of the right side of the expression. If you do not specify an ADDSTR, DELETEWHEN is based on the VAR specified.

For example:

```
ADDCONFIG
  VAR = 'FILES',
  ADDSTR = '60'
```

ADDWHEN — optional
ADDWHEN specifies when during processing the new line is added to the CONFIG.SYS.

Valid values of ADDWHEN are:

INSTALL (default)
Add the line during an installation action.

UPDATE (default)
Add the line during an update action.

ALWAYS
Add the line every time the package file is processed for any action.

DELETE
Add the line during a delete action.

NEVER
Never add the line.

RESTORE
Add the line during a restore action.

<user_defined_variable>
Add the line based on a variable that you define. You can use the same **logical expressions** [**Link to heading 6.59 on page 68**] that are available for WHEN and EXITWHEN entries for adding a line based on conditions that may occur.

For example, to add a line during installation or when a flag you have defined is true, you could use the following ADDWHEN:

```
ADDWHEN='(INSTALL || ("%MY_FLAG%"==TRUE))'
```

DELETEWHEN — optional
DELETEWHEN specifies when during processing the line is removed from the CONFIG.SYS.

Valid values of DELETEWHEN are:

NEVER (default)
Never delete the line.

ALWAYS
Delete the line every time the package file is processed for any action.

DELETE
Delete the line during a delete action.

DIREMPTY
Delete the line if the directory referenced by ADDSTR is empty.

INSTALL
Delete the line during an installation action.

RESTORE
Delete the line during a restore action.

UPDATE
Delete the line during an update action.

<user_defined_variable>
Delete the line based on a variable that you define.

You can use the same **logical expressions** [**Link to heading 6.59 on page 68**] that are available for WHEN and EXITWHEN entries for deleting a line based on conditions that may occur.

For example, to delete a line during installation or when a flag you have defined is true, you could use the following DELETEWHEN:

```
DELETEWHEN='(INSTALL || ("%MY_FLAG%"==TRUE))'
```

LOCATION — optional
LOCATION specifies where in the CONFIG.SYS file to add the new line.

Valid values of LOCATION are:

END (default)
Specifies that the line is added at the end of the file.

BEGIN
Specifies that the line is added at the beginning of the file.

FILESEARCHPOS — optional
Tells the installation and maintenance utility to add the new line before or after the FILESEARCHSTR.

Valid values of FILESEARCHPOS are:

AFTER (default)
Specifies to add the line after the FILESEARCHSTR.

BEFORE
Specifies to add the line before the FILESEARCHSTR.

FILESEARCHSTROCC — optional
FILESEARCHSTROCC specifies which occurrence of a multi-occurring string (FILESEARCHSTR) the installation and maintenance utility should use for the placement of the new line.

Valid values of FILESEARCHSTROCC are:

LAST (default)
Specifies that the last occurrence of the FILESEARCHSTR is used.

FIRST
Specifies that the first occurrence of the FILESEARCHSTR is used.

REPLACEOCC — optional
Specifies which occurrence of a line you want to replace.

Note: REPLACEOCC only has an affect when ADDWHEN evaluates to true. Use FILESEARCHSTR to specify the line you want to replace. The FILESEARCHSTR for a REPLACEOCC must be the entire line.

The installation and maintenance utility finds the occurrence of a line that matches the FILESEARCHSTR you specified and uses the REM command to remove it. The specified VAR and ADDSTR are added to the CONFIG.SYS after the removed line.

The product name is added to the line with the REM so that the user can see the effect of your product's installation on the CONFIG.SYS file.

REPLACEOCC has precedence over LOCATION and FILESEARCHPOS. If the installation utility does not find the FILESEARCHSTR used with a REPLACEOCC keyword, no line in the CONFIG.SYS is replaced and a new line is added based on how you specified the LOCATION keyword.

Valid values of REPLACEOCC are:

LAST
Specifies that the last occurrence of the FILESEARCHSTR is replaced.

ALL
Specifies that all occurrences of the FILESEARCHSTR are replaced.

FIRST
Specifies that the first occurrence of the FILESEARCHSTR is replaced.

TARGETDIR — optional
TARGETDIR specifies the drive and directory where CONFIG.SYS should be changed.

If you specify TARGETDIR as **<drive>:**, the current directory of <drive> is changed. If you specify TARGETDIR as **<drive>:**, the root directory of the <drive> is changed. You can also fully qualify the path you want to change.

Valid values of TARGETDIR are:

BOOT (default)
Specifies to the installation and maintenance utility that the CONFIG.SYS will be changed in the root directory of the drive where the operating system starts.

<specific path>
Specifies to the installation and maintenance utility that a specific CONFIG.SYS file will be changed. Multiple CONFIG.SYS files can be changed by placing one space between their paths.

For example, the following TARGETDIR specification would change the CONFIG.SYS in the operating system startup directory and in the root directory of the C: drive.

```
TARGETDIR = '%EPFIBOOTDRIVE% C:\'
```

UNIQUE — optional
UNIQUE specifies whether only one occurrence of a line (VAR=ADDSTR combination) is allowed in the CONFIG.SYS.

Valid values of UNIQUE are:

NO (default)
Multiple occurrences of the same line are allowed when adding a line.

YES
Only one occurrence of a line is allowed.

6.10 COMPONENT Entry Type

,,,,,,,

Autolink to Sibling Window

```
RES='&hres.' VPX='25%' VPY='bottom' VPCX='75%' VPCY='100%
```

- **Syntax** [**Link to heading 6.11 on page 44**]
- **Description** [**Link to heading 6.12 on page 44**]
- **Keywords** [**Link to heading 6.13 on page 45**]
- **Example** [**Link to heading 6.14 on page 46**]

6.11 COMPONENT Entry Type Syntax

```
COMPONENT
  NAME        ='<component>', (required)
  SIZE        ='<size of component in bytes>', (required)
  DESCRIPTION ='<description of component>', (optional)
  DISPLAY     ='<YES or NO>', (optional)
  ID          ='<identifier>', (optional)
  REQUIRES    ='<other components required>', (optional)
```

6.12 COMPONENT Entry Type Description

The COMPONENT entry type describes each part of your product. Breaking a product into components allows your product's user to install, delete, or update these parts independently.

The installation and maintenance utility transfers only files with FILE entries in components selected by the user. You should use the REQUIRES keyword for components that require other components.

COMPONENT entries in a package file show the beginning and end of the separately installable parts of your product. A component's information begins with a COMPONENT declaration and continues until the next COMPONENT entry is encountered (or until the package file ends). Components are installed in the order they occur in the package file.

If you use COMPONENT entries in a package file, the Install - directories window will contain a list box that displays all component names. This window prompts the user to select the components to install.

6.13 COMPONENT Entry Type Keywords

NAME — required

Name of a component of the product. It is a text string with a maximum length of 60 characters. The user sees this name on the Install - directories window. If you omit the NAME keyword, an error occurs when the installation and maintenance utility processes the package file. **Warning:** All component names specified in the package files for a given product must be unique. If you use duplicate component names you will not be able to delete or update the duplicate components using Software Installer.

SIZE — required

Disk space in bytes required to install the component selected from the Install - directories window. This value must be a string of not more than 9 digits. Commas and other separators are not allowed: *123456789* is valid, but *123.456,789* is not. When the user selects the **Check disk** push button from the Install - directories window, the installation and maintenance utility uses the size specified in this keyword to determine the amount of disk space needed to install the selected components.

The sum of all sizes specified in the SIZE keywords for all the COMPONENT entries should equal the disk space required to install the entire product, as specified in the SIZE keyword of the PACKAGE entry in the catalog file.

DESCRIPTION — optional

Provides the component description text that is to appear in the Component descriptions window when the **Description** push button in the Install - directories window is selected. The maximum allowable length of this text is 512 single-byte characters. If the DESCRIPTION keyword is omitted, the text ″No description available″ is displayed in the Component descriptions window.

DISPLAY — optional

Display indicator. It indicates whether the user sees the name of the component in the Install - directories window.

If not displayed, the component is installed only if another component which is selected by the user requires it by using the REQUIRES keyword.

Valid values are:

YES (default) Display the component

NO Do not display the component

ID — optional

Tag that uniquely identifies the component so other components can require it. It is a text string that can be up to eight characters long and does not include any spaces.

REQUIRES — optional

Specifies the other components that the installation and maintenance utility must install for this component to operate correctly. The REQUIRES keyword value is a string of one or more component IDs separated by spaces. If more than one ID is listed, the set must be enclosed in single quotes because of the space between the IDs. These IDs must match the values of the ID keywords of other COMPONENT entries. Only components in the same product can be required.

If one component requires a second component, the second component is automatically installed when the first component is installed even if the second component is not displayed (DISPLAY = NO). However, if the second component is displayed (DISPLAY = YES), the installation and maintenance utility displays a message to inform users that they must select the second component if the first is to be installed.

Following are examples of valid REQUIRES keyword values.

```
REQUIRES = COMP1
REQUIRES ='COMP2 COMP3'
```

6.14 COMPONENT Entry Type Example

All files after this COMPONENT entry in the package file but before the next COMPONENT entry are part of the same product component.

```
COMPONENT
  NAME='Promotional Inventory Control',
  ID = 'PROMO',
  * Display this component in the components list.
  DISPLAY = 'YES',
  * When this component is installed, also install the
  * 'Inventory Accounting' component defined above.
  REQUIRES = 'INVACCT',
  SIZE = '500000',
  * Define a description for this component.
  DESCRIPTION = 'Promotional Inventory Control manages the
  items in a store that are promotional or seasonal.'
```

6.15 DISK Entry Type

,,,,,,,

Autolink to Sibling Window

```
RES='&hres.' VPX='25%' VPY='bottom' VPCX='75%' VPCY='100%
```

- **Syntax** [**Link to heading 6.16 on page 46**]
- **Description** [**Link to heading 6.17 on page 46**]
- **Keywords** [**Link to heading 6.18 on page 47**]
- **Example** [**Link to heading 6.19 on page 47**]

6.16 DISK Entry Type Syntax

```
DISK
  NAME   ='<user friendly name>', (required)
  VOLUME='<volume label>' (conditionally required)
```

6.17 DISK Entry Type Description

The DISK entry type associates a user-friendly name to a disk. A package file that supports installation or update from a diskette must contain one DISK entry for each diskette used during the installation of the product.

For host and LAN installations, this entry type is ignored.

6.18 DISK Entry Type Keywords

NAME — required
The name that the user sees when the installation and maintenance utility prompts for the disk on the Insert disk window. This name should match the printed label that is attached to the disk. The NAME keyword is a text string with a maximum length of 60 characters.

VOLUME — conditionally required
The OS/2 volume label that identifies the disk. When the product is on more than one CD-ROM or diskette, the value of this keyword must match the volume label on one of the product disks. This keyword is a text string with a maximum length of 11 characters.

6.19 DISK Entry Type Example

You should define a user-friendly name for the diskette that contains the files to be installed.

```
DISK
  NAME='Software Installer Disk 1',
  * The files are on a diskette whose label is INSTALL1.
  VOLUME='INSTALL1'
```

6.20 EXIT Entry Type

,,,,,,,

Autolink to Sibling Window

```
RES='&hres.' VPX='25%' VPY='bottom' VPCX='75%' VPCY='100%
```


- **Syntax** [**Link to heading 6.21 on page 47**]
- **Description** [**Link to heading 6.22 on page 47**]
- **Keyword** [**Link to heading 6.23 on page 48**]
- **Example** [**Link to heading 6.24 on page 48**]

6.21 EXIT Entry Type Syntax

```
EXIT
  DLL=='<DLL file name>' (required)
```

6.22 EXIT Entry Type Description

An EXIT entry type specifies which DLL file is used for installation exits. The first EXIT entry in a package file is used until another EXIT entry is encountered.

If an EXIT entry does not appear in the package file before an installation exit is called, the default DLL file (EPFIEXTS.DLL) is used. EPFIEXTS.DLL is shipped with Software Installer and contains several useful **default installation exits**[**Link to heading 13.9 on page 89**].

If you do not fully qualify the DLL file name in the DLL keyword value, Software Installer searches for the DLL in the LIBPATH defined in the user's CONFIG.SYS. This search is done when the DLL file is loaded.

This entry does not start an installation exit. It specifies the DLL file that is used when an exit is called. You specify the actual start of an installation exit with the EXIT keyword in a FILE entry.

6.23 EXIT Entry Type Keyword

DLL — required
Workstation name of the DLL file that user exits are to use. It must be a valid OS/2 file name.

You should use double forward slashes as the directory separator in the fully qualified DLL file name. The double forward slashes are converted to a single backslash by the installation and maintenance utility.

Installation exits that are called continue to use the DLL file you specify in this keyword until another EXIT entry is encountered.

Installation variable[**Link to heading Chapter 17 on page 118**] substitution can be used in this keyword value.

6.24 EXIT Entry Type Example

```
EXIT
 DLL = EPFIEXTS
```

6.25 FILE Entry Type

,,,,,,,

Autolink to Sibling Window

```
RES='&hres.' VPX='25%' VPY='bottom' VPCX='75%' VPCY='100%
```

- **Syntax** [**Link to heading 6.26 on page 48**]
- **Description** [**Link to heading 6.27 on page 49**]
- **Keywords** [**Link to heading 6.28 on page 49**]
- **Example** [**Link to heading 6.29 on page 54**]

6.26 FILE Entry Type Syntax

```
FILE
  DATE           ='<date of source file>', (optional)
  DOWNLOAD       ='<download type>', (optional)
  EXIT           ='<installation exit>', (conditionally required)
  EXITIGNOREERR  ='<YES or NO>', (optional)
  EXITWHEN       ='<when to call the installation exit>', (optional)
  HIDDEN         ='<YES or NO>', (optional)
  NAME           ='<descriptive name of file>', (optional)
  PACKID         ='<PACKFILE ID>', (conditionally required)
  PWS            ='<PWS file name and extension>' (conditionally required)
  PWSPATH        ='<AUXn | FILE | PWSFILE | WORK>' (optional)
  READONLY       ='<YES or NO>', (optional)
  REPLACEINUSE   ='<D | I | R | U>', (optional)
  SIZE           ='<size of source file>', (optional)
  SOURCE         ='<name of source file>', (conditionally required)
  TIME           ='<time of source file>', (optional)
  UNPACK         ='<Yes , No, or unpack command to use>', (optional)
  VOLUME         ='<diskette volume>', (conditionally required)
  WHEN           ='<download conditions>' (optional)
```

6.27 FILE Entry Type Description

You can use package file FILE entries to transfer files, call **installation exits**[**Link to heading Chapter 13 on page 84**], or do both.

The FILE entry type specifies how a file is to be transferred from the source system to the workstation. You create one FILE entry for each product file that is to be transferred.

FILE entries that *only* call exits (do not transfer files) should use only the EXIT, EXITWHEN, and EXITIGNOREERR keywords.

The number of FILE entries in a package file (or set of package files, if the INCLUDE entry is used) determines the number of files that are transferred. When a user installs the product, the installation Progress window displays the number of FILE entries found in the package file.

The FILE entry contains keywords that give the name of the file on the source system, the name of the workstation file to transfer to, and additional file information like date, time, and other attributes.

6.28 FILE Entry Type Keywords

DATE — optional

Date of the file on the source system. It is in the YYMMDD format, where YY is the year, MM is the month, and DD is the day. This date should correspond to the date the file was compiled or created on the workstation.

If the WHEN keyword is set to OUTOFDATE, the value of this keyword is compared to the date of the existing workstation file to help determine whether to transfer the file. Only files more recent than an existing file of the same name are transferred when the WHEN keyword is set to OUTOFDATE.

The date of the workstation file is set to the date specified in this keyword when the file is transferred. Along with the time, the date is very useful when servicing a product because it uniquely identifies the service level of the file.

If the DATE keyword is not specified and the source is a host system, the DATE is set to the current date on the workstation. If the DATE keyword is not specified and the source is a **drive**[**Link to footnote, ID=drivfn on page 129**], the DATE is set to the date of the source file.

DOWNLOAD — optional
Type of transfer.

Valid values are:

STATIC
(default) Transfers the file in binary format.

DELETE
Deletes the workstation file, if it already exists. You can use this value to eliminate unused or obsolete files when a product is updated.

EXIT — conditionally required
Specifies the **installation exit**[**Link to heading 13.3 on page 85**] to call. An installation exit is not called if you omit this keyword.

An installation exit is called only if the file specified in the SOURCE keyword of the FILE entry is transferred or deleted (if DOWNLOAD equals DELETE).

If the FILE entry does not contain PWS or SOURCE keywords, but does contain an EXIT keyword, the installation exit is called. You can also use the EXITWHEN keyword to conditionally call the exit.

To pass information to installation exits, you can use **installation variables** [**Link to heading Chapter 17 on page 118**] for the value of this keyword.

You can use double forward slashes as the directory separator in the fully qualified file name of the exit. The double forward slashes are converted to a single backslash by the installation and maintenance utility. Fully specifying the DLL name eliminates the need for the DLL to be in a directory that exists in the LIBPATH statement of the CONFIG.SYS file.

EXITIGNOREERR — optional
Indicator of whether the installation and maintenance utility is to display an error message if an installation exit returns a nonzero result code or continue without displaying an error.

Valid values are:

YES Do not display a message, no matter what the result code is.

NO (default) Display an error message and stop the installation process if the user exit returns a nonzero result code.

See **Using Exits to Perform Product-Specific Tasks** [**Link to heading 13.1 on page 84**] for more about stopping processing.

EXITWHEN — optional
Specifies when the installation and maintenance utility will call the installation exit, if one is specified.

Logical expressions and conditional constants[**Link to heading 6.59 on page 68**] give you flexibility in specifying when the exit is called.

If this keyword is omitted, any exit specified will be called only during an installation or update.

HIDDEN — optional
Indicator of whether the workstation file is to have the OS/2 hidden attribute set after being transferred.

Valid values are:

YES Set the hidden attribute.

NO (Default) Do not set the hidden attribute.

NAME — optional

Descriptive name for the file being transferred. It is a text string with a maximum length of 32 characters. The installation and maintenance utility does not use this keyword, but you can use it to pass descriptive information to **installation exits.** [**Link to heading Chapter 25 on page 128**]

PACKID — conditionally required

Indicates that the file is contained in a file identified by a **PACKFILE**[**Link to heading 6.40 on page 58**] entry. The value of this keyword is an identifier that should match the value of the ID keyword in the corresponding PACKFILE entry. The identifier is a text string with a maximum of 8 characters.

Specifying this keyword in a FILE entry causes the SOURCE and VOLUME keywords to be ignored (if they exist). If a PACKFILE entry that matches the ID specified by this keyword is not found in the package files, an error message is displayed.

PWS — conditionally required

File name and extension of the transferred file on the workstation. The file is not transferred if you omit the PWS keyword.

If you specified the PWSPATH keyword in a FILE entry which has a PWS keyword, the file specified in the keyword must be fully qualified for the **DISKGEN /A option**[**Link to heading 4.2 on page 14**] to work correctly.

The workstation file name created by concatenating the value of the PWS keyword with the directory the user enters in the Install - directories window must not exceed the system limit. The maximum length is 128 characters.

If you specified the PWSPATH keyword in a FILE entry which has a PWS keyword, the file specified by PWS must be fully qualified for the Diskette Generator to work properly with the /A option.

Installation variable substitution [**Link to heading 13.4 on page 86**] can be used in this keyword value to help specify the workstation destination.

You can specify subdirectories that precede the file name and extension. If you specify subdirectories, the file is transferred to these subdirectories under the directory specified by the PWSPATH keyword. The subdirectories are created if they do not already exist.

You can use a forward slash (/) or a backslash () as a directory separator in the PWS keyword. The installation and maintenance utility translates forward slashes to backslashes. We recommend that you use a forward slash because it has the same hexadecimal value regardless of the code page.

Although Software Installer supports High Performance File System (HPFS) names, we do not recommend that you use them unless your product requires HPFS. If you use HPFS names, a user who does not have HPFS receives an error message during installation because the HPFS directories are not valid on the user's system.

Examples

```
PWS = ' FILE1.EXE'
PWS = ' D:/FILE2.DLL'
PWS = '%TEMPDIR%/FILE1.EXE'
```

PWSPATH — optional

Specifies whether the file is placed in the directory specified by the file entry field, the data entry field, or the auxiliary entry field.

The default names of the entry fields for directories on the Install - directories window are **File directory** (file entry field), **Data directory** (data entry field), and **Auxiliary directory n** (auxiliary entry field) where **n** is the digit 1 through 18. You can change the default names and default directories of these entry fields by using the label keywords in the **PATH entry type**[**Link to heading 6.45 on page 60**]. The user can also change the directory in the entry field beside the respective directory name during installation.

Valid values for PWSPATH are:

AUXn Places the file in the directory specified in the auxiliary entry field on the Install - directories window. The particular auxiliary directory value corresponds to the value specified for that AUXn keyword of the PATH entry.

If you specify a value for this keyword that does not match a corresponding keyword in the PATH entry, the default is used. For example, if you specify AUX3 as the value of PWSPATH, but do not specify an AUX3 keyword for the PATH entry, the directory specified for the file entry field is used.

FILE (default) Places the file in the directory specified in the file entry field in the Install - directories window. The file entry field value corresponds to the value specified for the FILE keyword of the PATH entry.

PWSFILE Places the file in the directory specified in the fully qualified file name in the PWS keyword.

WORK Places the file in the directory specified in the data entry field in the Install - directories window. The data entry field value corresponds to the value specified for the WORK keyword of the PATH entry.

READONLY — optional

Indicator of whether the workstation file is to have the OS/2 read-only attribute set after being transferred.

Valid values are:

YES Set the read-only attribute.

NO (Default) Do not set the read-only attribute.

REPLACEINUSE — optional

Indicates when the user wants the **in-use copy of a file replaced**[**Link to heading Chapter 11 on page 82**]. If this keyword is not used, the in-use copy of a file is not replaced. Valid values are:

'D' Replace on delete processing
'I' Replace on installation processing
'U' Replace on update processing
'R' Replace on restore processing

To specify multiple values, enclose all values in single quotes. For instance:

REPLACEINUSE='I U'

If you use REPLACEINUSE, your documentation should tell the user to restart the workstation operating system without deleting anything from the command line. This restart enables the installation and maintenance utility to finish the deferred processing of files that were in use. If a user does not restart the operating system when told, the CONFIG.SYS file may have a PROTSHELL statement that points to an installation shell instead of the Workplace Shell.

SIZE — optional

Size of the file. This value must be a string of not more than 9 digits. Commas and other separators are not allowed: *200000000* is valid, but *200,000,000* is not.

The installation and maintenance utility does not use this keyword. However, it can be used to pass information to installation exits. For additional information, see **installation variables.** [**Link to heading Chapter 17 on page 118**]

SOURCE — conditionally required

Location and file name of the file to be transferred. The file is not transferred if you omit this keyword. The maximum length is 255 characters.

You must use the following format for this keyword.

SOURCE = 'MVS: mvsname; VM: vmname; VSE: vsename; DRIVE: drivename'

This format allows you to create one package file that works for all source systems. You can omit any system which is not used as a source for installation of your product.

Specify the higher level data set qualifiers in the HOSTLOC keyword in the catalog file. If a HOSTLOC keyword is specified, then the value of HOSTLOC is concatenated in front of the SOURCE keyword value to obtain the fully qualified source file name. For additional information about the HOSTLOC keyword, see Keywords in the **PACKAGE**[**Link to heading 5.14 on page 22**] entry type section.

Valid values are:

mvsname Data set and member name on an MVS system.

For example, if PROJ.GROUP.TYPE(MEMBER) is the name of a member in the source system, the SOURCE keyword of the FILE entry that transfers this file should be MVS: TYPE(MEMBER). You can then set the HOSTLOC keyword in the catalog file to PROJ.GROUP..

vmname File name of the file on a VM system.

When you write a package file for VM, you specify a file name, file type, and file mode in the SOURCE keyword. The file mode should be an asterisk (*); for example, VM: FILENAME FILETYPE *. This specification allows users to link to the minidisk containing the files as any file mode. The installation and maintenance utility uses a normal CMS search order to find the source file.

Shared file directories in VM are supported only if the directory is accessed as a file mode.

vsename File name of the file on a VSE system.

When you write a package file for VSE, you specify a member name and member type in the SOURCE keyword. The SOURCE keyword should contain only the last file name qualifier followed by the member type. Specify the higher level data set qualifiers in the HOSTLOC keyword in the catalog file.

For example, if LIB.SUBLIB.MEMBERNAME.MEMBERTYPE is the name of a member in the source system, the SOURCE keyword of the FILE entry that transfers this file should be VSE: MEMBERNAME.MEMBERTYPE. You should set the HOSTLOC keyword in the catalog file to LIB.SUBLIB..

drivename Name of the file when the source is an OS/2 local drive or a LAN drive. This is a valid OS/2 file name, including directories but *not* including a drive. The drive is obtained from the Open drive catalog window.

TIME — optional

Time of the file on the source system. It is in the HHMM format, where HH is the hour and MM is the minute. If the WHEN keyword is set to OUTOFDATE, the value of this keyword is compared to the time of the existing workstation file to help determine whether to transfer the file.

The time of the workstation file is set to the time specified in this keyword when the file is transferred. Along with the date, the time is very useful when servicing a product because it uniquely identifies the service level of the file.

If the TIME keyword is not specified and the source is a host system, the TIME is set to the current time on the workstation. If the TIME keyword is not specified and the source is a drive, the TIME is set to the time of the source file.

UNPACK — optional

Specifies whether or not to unpack a compressed file and which unpack utility to use.

Valid values are:

NO (default) Do not unpack the file.

YES Unpack the file.

Note: UNPACK should be set to YES only if the file specified by the SOURCE keyword value has been packed using EPFIPACK.EXE.

<utility command>

Specifies the unpack utility to be used.

Any entry other than YES or NO is recognized as an instruction for performing the unpacking. **Installation variable substitution** [**Link to heading 13.4 on page 86**] can be used in this keyword value to specify the full path of the unpack utility and specify the parameters for the utility. **Installation variables** [**Link to heading Chapter 17 on page 118**] EPFIFILEDIR, EPFICURUPD, EPFICURUPDIR, and EPFICURUPS are particularly useful in specifying the unpack command using your own utility. For example:

```
PACKFILE
  .
  .
  UNPACK='%EPFIFILEDIR%//LOADRAM.EXE %EPFICURUPS% %EPFICURUPDIR%'
```

specifies using a LOADRAM.EXE that has been previously transferred to the directory in the EPFIFILEDIR variable to unpack the file specified by the EPFICURUPS variable. The file is unpacked into the directory specified by the EPFICURUPDIR variable. The double forward slashes are converted to a single backslash by the installation and maintenance utility.

If the utility you use does not overwrite existing files as the default, be sure to specify the parameter that will result in overwriting.

VOLUME — conditionally required

Specifies which disk the file is on when there is more than one disk. The VOLUME keyword is ignored if the catalog was opened from a source that is not a removable media (CD-ROM or diskette) drive (MVS, VM, VSE or LAN).

This value should match the VOLUME keyword of a DISK entry. As each FILE entry is processed during a disk install or update the VOLUME keyword is checked against the OS/2 volume label of the disk in the drive before attempting to copy the file. If the keyword value and the OS/2 volume label of the disk do not match, the Insert disk window is displayed. The name of the disk prompted for on this window is determined by the NAME keyword of the corresponding **DISK entry**[**Link to heading 6.15 on page 46**]. If there is not a DISK entry that matches the VOLUME keyword for a file, the user is prompted by the actual volume label instead of a user-friendly name.

WHEN — optional

Conditions under which the file is to be transferred or deleted (if DOWNLOAD equals DELETE).

Logical expressions and conditional constants[**Link to heading 6.59 on page 68**] give you flexibility in specifying the conditions for transferring or deleting the file.

If the WHEN keyword is omitted, the default condition for processing is OUTOFDATE. In this case Software Installer will transfer or delete the file only if the date and time of the existing workstation file are earlier than the date and time specified in the DATE and TIME keywords. The file will also be transferred if it does not exist on the workstation.

6.29 FILE Entry Type Example

```
FILE
  * Install this file when the file on the workstation is
  * old or the file does not exist on the workstation.
  WHEN = 'OUTOFDATE',
  * Set the date of the file to January 28, 1992.
  DATE = '920128',
  * Set the time of the file to 08:30am.
  TIME = '0830',
  * Install this file in the WORK directory.
  PWSPATH = 'WORK',
  * Because the file was packed using EPFIPACK,
  * unpack it using EPFIUPCK.
  UNPACK = 'YES',
  * Define the possible locations of the packed file.
  SOURCE = 'MVS: FILES(INVACDLL);
            VM: INVACDLL DLLBIN *;
            VSE: INVACDLL.DLLBIN;
            DRIVE: INVACDLL.DL_',
  * Define which diskette the file exists on.
  VOLUME = 'INSTALL1',
  * Define the name given to this file on the workstation.
  PWS = 'INVENTACDLL',
  * Run the exit when this file is installed.
  EXITWHEN = 'INSTALL',
  * Ignore all errors from the exit.
  EXITIGNOREERR = 'YES',
  * Add the WORK directory to the LIBPATH installation variable.
  EXIT = 'UPDATECONFIG LIBPATH=%EPFIWORKDIR%;'
```

6.30 INCLUDE Entry Type

,,,,,,,

Autolink to Sibling Window

```
RES='&hres.' VPX='25%' VPY='bottom' VPCX='75%' VPCY='100%
```


6.31 INCLUDE Entry Type Syntax

```
INCLUDE
  NAME=='MVS: mvsname; VM: vmname; VSE: vsename; DRIVE: drivename'> (required)
```

6.32 INCLUDE Entry Type Description

The INCLUDE entry type causes another package file to be included when the package file is processed.

Note: You should not have more than five levels of included package files.

The following represents the maximum permitted five levels of package inclusion:

```
LEVEL1.PKG includes LEVEL2.PKG
  LEVEL2.PKG includes LEVEL3.PKG
    LEVEL3.PKG includes LEVEL4.PKG
      LEVEL4.PKG includes LEVEL5.PKG
```

However, you can include several catalog files from one level.

The capability of including other package files is useful for packaging products that are translated into multiple national languages. You can create multiple package files for such products. Create one for the base or nontranslatable files, and one for the translatable or machine-readable information (MRI) files of each national language. Each MRI package file can include the base package file, which eliminates the need to specify the base files separately for each language.

If the HOSTLOC keyword is specified in the catalog file, the value of HOSTLOC is concatenated with the INCLUDE entry type's NAME keyword value to obtain the fully qualified file name.

6.33 INCLUDE Entry Type Keyword

NAME — required
Source system name of the package file to be included. The maximum length is 255 characters.

Valid values are:

mvsname	Data set name on MVS
vmname	File name on VM
vsename	File name on VSE
drivename	File name on a drive

6.34 INCLUDE Entry Type Example

```
* Define the package file to include with the current package file.
* Be sure to have a catalog file PACKAGE entry HOSTLOC keyword for
* each source system location.
INCLUDE
  NAME='MVS: TYPE(INCLPKG); VM: INCLPKG PACKAGE *; DRIVE: INCLPKG.PKG; VSE: INCLCAT.PACKAGE'
```

6.35 OPTIONS Entry Type

,,,,,,,

Autolink to Sibling Window

```
RES='&hres.' VPX='25%' VPY='bottom' VPCX='75%' VPCY='100%
```

- **Syntax** [**Link to heading 6.36 on page 57**]
- **Description** [**Link to heading 6.37 on page 57**]
- **Keywords** [**Link to heading 6.38 on page 57**]
- **Example** [**Link to heading 6.39 on page 58**]

6.36 OPTIONS Entry Type Syntax

```
OPTIONS
  SUCCESSDELMSG =<'text for successful delete message'>, (optional)
  SUCCESSINSMSG =<'text for successful install message'>, (optional)
  SUCCESSRESMSG =<'text for successful restore message'>, (optional)
  SUCCESSUPDMSG =<'text for successful update message'> (optional)
```

6.37 OPTIONS Entry Type Description

The OPTIONS entry type allows you to specify the text of messages displayed to the user after successful install, update, restore, and delete actions.

You specify the success message for an action by using text enclosed in single quotes following the keyword. There is a keyword for each action.

All keywords can contain a text string with a maximum length of 255 bytes including any variable substitution. If you use variable substitution, the total message length can be a maximum of 1024 bytes.

Remember that proportional fonts may result in truncation of your desired message.

If the same keyword occurs more than once in a package file, the installation and maintenance utility uses the last keyword.

6.38 OPTIONS Entry Type Keywords

SUCCESSDELMSG — optional
Specifies the message for successful completion of a delete action.

If you do not specify a message text, the default is:

The selected components of <product name> have been successfully deleted.

SUCCESSINSMSG — optional
Specifies the message for successful completion of an install action.

If you do not specify a message text, the default is:

The requested components of <product name> are successfully installed.
Elapsed time was <seconds>.
Before starting the product, you might need to start the system again to activate any new paths added to your CONFIG.SYS file.

SUCCESSRESMSG — optional
Specifies the message for successful completion of a restore action.

If you do not specify a message text, the default is:

<product name> is successfully restored to the previous service level.

SUCCESSUPDMSG — optional
Specifies the message for successful completion of an update action.

If you do not specify a message text, the default is:

<product name> is successfully updated.
Elapsed time was <seconds>.
Before starting the product, you might need to start the system again to activate any new paths added to your CONFIG.SYS file.

6.39 OPTIONS Entry Type Example

```
OPTIONS
  SUCCESSINSMSG ='The components for XYZ Product Version 1.2 have been successfully
  installed.  The installation process updated your CONFIG.SYS.  You must restart your
  system to use XYZ Product.'
```

6.40 PACKFILE Entry Type

,,,,,,,

Autolink to Sibling Window

```
RES='&hres.' VPX='25%' VPY='bottom' VPCX='75%' VPCY='100%
```

- **Syntax** [**Link to heading 6.41 on page 58**]
- **Description** [**Link to heading 6.42 on page 58**]
- **Keywords** [**Link to heading 6.43 on page 59**]
- **Example** [**Link to heading 6.44 on page 59**]

6.41 PACKFILE Entry Type Syntax

```
PACKFILE
  ID        ='<identifier>', (required)
  SOURCE    ='<name of the file on the host or drive>', (required)
  EXIT      ='<installation exit>', (optional)
  EXITWHEN  ='<when to call the installation exit>', (optional)
  UNPACK    ='<Yes, No or name of utility to call>', (optional)
  VOLUME    ='<diskette volume label>' (conditionally required)
```

6.42 PACKFILE Entry Type Description

The PACKFILE entry type is used if you are using multiple **file compression** [**Link to heading 10.1 on page 80**] in packaging your product. A PACKFILE entry should exist for each packed file that was made from more than one file.

The PACKFILE keyword causes the compressed file to be unpacked. The individual file entries indicate that these files are found in the packed file named in the PACKID keyword.

6.43 PACKFILE Entry Type Keywords

ID — required
An identifier that identifies the PACKFILE entry. This identifier is referenced by the PACKID keyword of the FILE entry to indicate that a file is in this PACKFILE. The identifier is a text string with a maximum length of 8 characters.

SOURCE — required
The name of the compressed (packed) file in the source system or **drive**[**Link to footnote, ID=drivfn on page 129**]. This keyword tells the installation and maintenance utility where to find the file.

If the PACKAGE entry HOSTLOC keyword is specified in the catalog file, the value of HOSTLOC is concatenated with this keyword value to obtain the fully qualified file name.

You must use the following format for this keyword.

SOURCE = 'MVS: mvsname; VM: vmname; VSE: vsename; DRIVE: drivename'

EXIT — optional
Specifies an installation exit to call after the compressed file is transferred. The value of this keyword follows the same rules as the EXIT keyword of the FILE entry. **Installation variable**[**Link to heading Chapter 17 on page 118**] substitution can be used in this keyword value.

EXITWHEN — optional
Instructs the installation and maintenance utility when to call the installation exit, if one is specified.

Logical expressions and conditional constants[**Link to heading 6.59 on page 68**] give you flexibility in specifying when the exit is called.

If this keyword is omitted, any exit specified will be called only during an installation or update.

UNPACK — optional
Specifies whether or not to unpack the file after transferring it. Valid values are YES, NO, or the name of the utility to be used. If YES is specified the EPFIPACK utility is used. If this keyword is omitted, the default value is YES. If the product is packaged so that multiple files are compressed into one file, the PACKFILE entry specifies the name and location of the packed file.

If the file was packed using your own utility, unpack it with the corresponding unpacking utility by using the name of your unpacking utility in the UNPACK keyword.

If the utility you use does not overwrite existing files as the default, be sure to specify the parameter that will result in overwriting. **Installation variable**[**Link to heading Chapter 17 on page 118**] substitution is useful in this keyword value. For example, UNPACK='PKUNZIP.EXE -o %EPFICURUPS% %EPFICURUPDIR%' would cause the PKUNZIP utility to be used to unpack the file specified by the EPFICURUPS installation variable into the directory specified by the EPFICURUPDIR installation variable.

You should use the -o option for PKUNZIP to overwrite existing files. If you use another unpacking utility, you should also use any option that specifies overwriting existing files.

VOLUME — conditionally required
Specifies which disk the PACKFILE file is on. This value should match the VOLUME keyword of a DISK entry and is ignored if the files are not being installed from a removable media source.

6.44 PACKFILE Entry Type Example

```
PACKFILE
  SOURCE = ' MVS: FILES(PERSHPAK);
             VM: PERSHPAK PAKBIN *;
             VSE: PERSHPAK.PAKBIN;
             DRIVE: PERSHPAK.PA_' ,
  ID = ' PERSH'
  UNPACK = ' PKUNZIP.EXE -o %EPFICURUPS% %EPFICURUPDIR%'
```

6.45 PATH Entry Type

,,,,,,,

Autolink to Sibling Window

```
RES='&hres.' VPX='25%' VPY='bottom' VPCX='75%' VPCY='100%
```

- **Syntax** [**Link to heading 6.46 on page 60**]
- **Description** [**Link to heading 6.47 on page 60**]
- **Keywords** [**Link to heading 6.48 on page 61**]
- **Example** [**Link to heading 6.49 on page 62**]

6.46 PATH Entry Type Syntax

```
PATH
  AUXn         ='<default auxiliary directory>', (optional)
  AUXnLABEL    ='<maximum 30 characters>', (optional)
  DESCRIPTION  ='<maximum 512 characters>', (optional)
  DRIVEONLY    ='<directory where only the drive can be entered>', (optional)
  FILE         ='<default installation directory>', (optional)
  FILELABEL    ='<maximum 30 characters>', (optional)
  WORK         ='<default data directory>', (optional)
  WORKLABEL    ='<maximum 30 characters>' (optional)
```

6.47 PATH Entry Type Description

The PATH entry type specifies the paths used for transferring all files. The installation and maintenance utility displays the paths you specify in this entry to the user in the Install - directories window. The user can override any paths in this window.

The PATH entry is not required. If no PATH entry exists in a package file, the Install - directories window does not display a default path. When no PATH entry is specified, only a **File directory** is prompted for. The user must enter the directory where the product should be installed.

If the same keyword occurs more than once in a package file, the installation and maintenance utility uses the last keyword.

You can use a forward slash (/) or a backslash () as a directory separator in any keyword value in a PATH entry. The installation and maintenance utility translates forward slashes to backslashes. It is recommended that you use a forward slash because it has the same hexadecimal value regardless of the code page.

For example, both of the following are valid directory specifications.

```
PATH
  FILE =D:/WP/SYSTEM
  WORK=C:\WP\SYSTEM
```

Although Software Installer supports High Performance File System (HPFS) names, we do not recommend that you use them unless your product requires HPFS. If you use HPFS names, a user who does not have HPFS receives an error message during installation because the HPFS directory names are not valid on the user's system.

6.48 PATH Entry Type Keywords

AUXn — optional
Specifies the path the user sees in the entry field named by the AUXnLABEL keyword in the Install - directories window, where **n** is any number between 1 and 18. (Thus there are 18 valid keywords: AUX1, AUX2,...AUX18).

AUXnLABEL — optional
Specifies the name that appears beside the AUXn entry field in the Install - directories window. The keyword can be a maximum of 30 characters in length. The maximum number of characters might not fit in the space allowed, depending on the size of the default system. If this keyword is omitted, then the default label is **Auxiliary directory n:**, where n is 1 -18. If the corresponding AUXn keyword is omitted, the AUXnLABEL keyword is ignored.

DESCRIPTION — optional
Specifies the description that appears above the directory entry fields on the Install - directories window. The description can be a maximum of 512 characters in length. Four lines of text are displayed. Fewer than the maximum number of characters might fit on these four lines because of the default system font. If the keyword is omitted, a prompting message is displayed. Which message is displayed depends on whether components are installed or a product is partially installed.

DRIVEONLY — optional
Specifies whether any of the directory entry fields on the Install - directories window allow only drives to be entered. If an invalid path is entered, an error message is displayed when the **Install** push button is selected. Valid values for this keyword are any combination of FILE, WORK, or AUXn. Separate these values from each other with at least one blank space and enclose them with singles quotes. If this keyword is omitted, users must enter fully qualified paths in the Install - directories window. For example:

```
DRIVEONLY = ' AUX1 AUX2 AUX3'
```

FILE — optional
Specifies the path the user sees in the entry field named by the FILELABEL keyword in the Install - directories window. This keyword must be a valid, fully qualified OS/2 directory and must specify a drive. The path specified in this keyword is created if it does not already exist. If you omit the FILE keyword, the **FILE** directory line will show a blank entry field, instead of a default path.

FILELABEL — optional
Specifies the name that appears beside the FILE entry field in the Install - directories window. This keyword can be up to 30 characters in length. Fewer than the maximum number of characters might fit in the space allowed because of the default system font. If the FILELABEL keyword is omitted, **File directory** is the default.

WORK — optional
Specifies the default path the user sees in the entry field of the **Data directory** line in the Install - directories list. It must be a valid, fully qualified OS/2 directory and must specify a drive. If you omit the WORK keyword, the **Data directory** line is omitted from the directories list.

WORKLABEL — optional
Specifies the name that appears beside the **Data directory** field in the Install - directories window. This keyword can be up to 30 characters in length. Fewer than the maximum number of characters might fit in the space allowed because of the default system font. If this keyword is omitted, **Data directory:** is used as the default.

6.49 PATH Entry Type Example

This example of a PATH entry causes Software Installer to prompt the user installing the product for seven directories.

```
PATH
  FILE = 'C:/SUPERMKT/INVNTRY/EXE',
  FILELABEL = 'Executable directory:',
  WORK = 'C:/SUPERMKT/INVNTRY/DLL',
  WORKLABEL = 'DLL directory:',
  AUX1 = 'C:/SUPERMKT/INVNTRY/DOC',
  AUX1LABEL = 'Documentation directory:',
  AUX2 = 'C:/SUPERMKT/INVNTRY/DATA/PROMO',
  AUX2LABEL = 'Promotional data:',
  AUX3 = 'C:/SUPERMKT/INVNTRY/DATA/PERISH',
  AUX3LABEL = 'Perishables data: ',
  AUX4 = 'C:/SUPERMKT/INVNTRY/DATA/NOPERISH',
  AUX4LABEL = 'Non-perishables data:',
  AUX5 = 'C:/SUPERMKT/INVNTRY/DATA/NONFOOD',
  AUX5LABEL = 'Non-food data:'
```

6.50 SERVICELEVEL Entry Type

,,,,,,,

Autolink to Sibling Window

```
RES='&hres.' VPX='25%' VPY='bottom' VPCX='75%' VPCY='100%
```

- **Syntax** [**Link to heading 6.51 on page 62**]
- **Description** [**Link to heading 6.52 on page 63**]
- **Keyword** [**Link to heading 6.53 on page 63**]
- **Example** [**Link to heading 6.54 on page 63**]

6.51 SERVICELEVEL Entry Type Syntax

```
SERVICELEVEL
  LEVEL ='<service level>' (required)
```

6.52 SERVICELEVEL Entry Type Description

The SERVICELEVEL entry type specifies the level of service of the product files that you list in the package file. The installation and maintenance utility uses the service level to keep track of the corrections that have been applied to a product. The service levels applied to the selected product are displayed in the Service level window when the user selects the **Service level** push button on the Product status window.

6.53 SERVICELEVEL Entry Type Keyword

LEVEL — required

Identifies the level of corrections applied to a product. It is a text string that can be up to 60 characters long. When the product is initially shipped, you should set this field to 000000.

Each time a product update is shipped, a new package file must be included. The LEVEL keyword for the new package file should be set to the identification number of the correction level being shipped. Users see the service level in windows throughout the installation program. The installation and maintenance utility uses the LEVEL keyword to identify the last product level installed.

6.54 SERVICELEVEL Entry Type Example

```
SERVICELEVEL
  * Set the service level to a value that indicates
  * the release level of the product files.
  LEVEL = '0000000'
```

6.55 UPDATECONFIG Entry Type

,,,,,,,

Autolink to Sibling Window

```
RES='&hres.' VPX='25%' VPY='bottom' VPCX='75%' VPCY='100%
```

- **Syntax** [**Link to heading 6.56 on page 63**]
- **Description** [**Link to heading 6.57 on page 64**]
- **Keywords** [**Link to heading 6.58 on page 65**]

6.56 UPDATECONFIG Entry Type Syntax

```
UPDATECONFIG
  ADDSTR          ='<line to process>', (required)
  VAR             ='<lValue>', (required)
  ADDWHEN         ='<INSTALL|UPDATE|ALWAYS|NEVER|RESTORE|DELETE|<user_defined_variable>>', (optional)
  DELETEWHEN      ='<NEVER|ALWAYS|DIREMPTY|INSTALL|UPDATE|RESTORE|DELETE|<user_defined_variable>>', (optional)
  LINELOCATION    ='<END|BEGIN>', (optional)
  LINESEARCHPOS   ='<AFTER|BEFORE>', (optional)
  LINESEARCHSTR   ='<string within a line>', (optional)
  FILESEARCHPOS   ='<AFTER|BEFORE>', (optional)
  FILESEARCHSTR   ='<search string>', (optional)
  FILESEARCHSTROCC='<LAST|FIRST>', (optional)
  TARGETDIR       ='<BOOT|<path>>', (optional)
  VAROCC          ='<LAST|FIRST>', (optional)
  VARVALUE        ='<variable value> (optional)
```

6.57 UPDATECONFIG Entry Type Description

You can use the UPDATECONFIG entry type to:

- Add strings to lines in the user's CONFIG.SYS file
- Add the new strings before or after a target string that you specify
- Delete strings from lines in the user's CONFIG.SYS file
- Specify which paths to change the CONFIG.SYS files in

If your product's installation requires modification of the user's CONFIG.SYS file and the user does not select the Update CONFIG.SYS check box, an updated configuration file named CONFIG.ADD is copied from memory to the directory containing the CONFIG.SYS that needs to be modified.

If the user does select the Update CONFIG.SYS check box, the installation utility updates the user's configuration file and copies the old CONFIG.SYS to CONFIG.BAK in the directory where the CONFIG.SYS was modified.

The installation and maintenance utility uses the following precedence in positioning the change in CONFIG.SYS. (Default values are underlined.)

1. Find an occurrence of VAR containing VARVALUE in its rValue
2. Find the FILESEARCHSTR
   - Uses FILESEARCHSTROCC (LAST|FIRST) to find the FILESEARCHSTR
   - Uses FILESEARCHPOS (AFTER|BEFORE) to find the VAR relative to the FILESEARCHSTR
3. Uses VAROCC (LAST|FIRST) if FILESEARCHSTR is not found

The installation and maintenance utility uses the following precedence in locating where to make the change in the line.

1. Searches for LINESEARCHSTR
   - Uses LINESEARCHPOS (AFTER|BEFORE)
2. Uses LINELOCATION (LAST|FIRST)

6.58 UPDATECONFIG Entry Type Keywords

You can use the following keywords with the UPDATECONFIG entry type to modify the user's CONFIG.SYS file during installation and maintenance processing.

Keyword values are strings with a maximum length of 1024 characters.

ADDSTR — required
ADDSTR specifies the string you want added to a line in the CONFIG.SYS.

For example:

```
UPDATECONFIG
  VAR = 'SET PATH',
  ADDSTR = 'C:\IBM\OS2\BIN'
```

VAR — required
VAR specifies the lValue (left side of the equals sign) which identifies the line you want to change in the CONFIG.SYS.

For example:

```
UPDATECONFIG
  VAR = 'SET PATH'
```

ADDWHEN — optional
ADDWHEN specifies when during processing the ADDSTR will be added to the specified VAR in the CONFIG.SYS.

Valid values of ADDWHEN are:

INSTALL (default)
Add the string during an installation action.

UPDATE (default)
Add the string during an update action.

ALWAYS
Add the string every time the package file is processed for any action.

DELETE
Add the string during a delete action.

NEVER
Never add the string.

RESTORE
Add the string during a restore action.

<user_defined_variable>
Add the string based on a variable that you define.

You can use the same **logical expressions** [**Link to heading 6.59 on page 68**] that are available for WHEN and EXITWHEN entries to base changes on conditions that may occur.

For example, to add a string during installation or when a flag you have defined is true, you could use the following ADDWHEN keyword:

```
ADDWHEN='(INSTALL || ("%MY_FLAG%"==TRUE))'
```

DELETEWHEN — optional
DELETEWHEN specifies when during processing the ADDSTR will be deleted from the specified VAR in the CONFIG.SYS.

Valid values of DELETEWHEN are:

NEVER (default)
Never delete the string.

ALWAYS
Delete the string every time the package file is processed for any action.

DELETE
Delete the string during a delete action.

DIREMPTY
Delete the string if the directory referenced by ADDSTR is empty.

INSTALL
Delete the string during an installation action.

RESTORE
Delete the string during a restore action.

UPDATE
Delete the string during an update action.

<user_defined_variable>
Delete the string based on a variable that you define. You can use **logical expressions**[**Link to heading 6.59 on page 68**] for deleting string base on conditions that may occur.

For example, to delete a string during installation or when a flag you have defined is true, you could use the following DELETEWHEN:

```
DELETEWHEN='(INSTALL || ("%MY_FLAG%"==TRUE))'
```

LINESEARCHPOS — optional
LINESEARCHPOS specifies the position within a line relative to the LINESEARCHSTR where the ADDSTR should be added.

AFTER (default)
Specifies that the ADDSTR is added after the LINESEARCHSTR

BEFORE
Specifies that the ADDSTR is added before the LINESEARCHSTR

LINESEARCHSTR — optional
LINESEARCHSTR specifies the string to use in searching within a line.

The ADDSTR is placed within the line relative to the LINESEARCHSTR. The first matching string within the line is used. You should try to use a large enough string to uniquely position the addition within the line.

LINELOCATION — optional
LINELOCATION specifies where in the line to add the string.

Valid values of LINELOCATION are:

END (default)
Specifies that the ADDSTR is added at the end of the line.

BEGIN
Specifies that the ADDSTR is added at the beginning of the line.

FILESEARCHPOS — optional
Specifies the position of a VAR relative to the FILESEARCHSTR

Valid values are:

AFTER (default)
The first occurrence of VAR after the string specified by FILESEARCHSTR.

BEFORE
The first occurrence of VAR before the string specified by FILESEARCHSTR.

FILESEARCHSTR — optional
FILESEARCHSTR specifies a string in a line you specify to find the VAR you want modified.

FILESEARCHSTROCC — optional
FILESEARCHSTROCC specifies which occurrence of a multi-occurring FILESEARCHSTR the installation and maintenance utility should use in finding the line to be changed.

LAST (default)
Specifies that the last occurrence is used.

FIRST
Specifies that the first occurrence is used.

TARGETDIR — optional
TARGETDIR specifies the drive and directory where CONFIG.SYS should be changed.

If you specify TARGETDIR as **<drive>:**, the current directory of <drive> is changed. If you specify TARGETDIR as **<drive>:**, the root directory of the <drive> is changed. You can also fully qualify the path you want to change.

Valid values are:

BOOT (default)
Specifies to the installation and maintenance utility that the CONFIG.SYS in the root directory of the drive where the operating system starts will be changed.

<specific path>
Specifies to the installation and maintenance utility that a specific CONFIG.SYS file will be changed. Multiple CONFIG.SYS files can be changed by placing one space between their paths.

For example, the following TARGETDIR specification would change the CONFIG.SYS in the operating system startup directory and in the root directory of the C: drive.

```
TARGETDIR = '%EPFIBOOTDRIVE% C:\'
```

VAROCC — optional
VAROCC specifies which occurrence of VAR you want to change.

Valid values are:

LAST (default)
Specifies that the last occurrence of a line with the specified VAR should be changed.

FIRST
Specifies that the first occurrence of a line with the specified VAR should be changed.

VARVALUE — optional
VARVALUE specifies a value of a VAR you want to find.

You can use a full or partial specification of a string on the right side of an expression you want to change.

6.59 Logical Expressions for WHEN and EXITWHEN

Software Installer allows you to use logical expressions in the WHEN and EXITWHEN keyword values of the FILE entry type in a package file.

Logical expressions are made up of operators and operands. Operands can themselves be expressions, installation variables (in the format "%var%"), numeric constants, or string constants. Numeric constants are constants that are entirely made up of digits, for example 123, 124235, or -234. String constants are constants that have at least one character that is not a digit or that are enclosed in double quotes ("). If the string constant contains a space or is a variable substitution, then it must be enclosed in double quotes. Logical expressions must evaluate to TRUE or FALSE.

For example:

```
WHEN= OUTOFDATE && "%TRANSFILE%"==YES'
WHEN=(123 > 789) && (("%complete%"=="done") || ("%complete%"=="almost done"))'
WHEN="%count%" > 10'
EXITWHEN = ' INSTALL && "%DOEXIT%"==YES'
```

The supported operators in WHEN and EXITWHEN expressions are:

= = Checks for equivalence. Returns TRUE if the operands are equivalent, FALSE otherwise.

! = Checks for non-equivalence. Returns FALSE if the operands are equivalent, TRUE otherwise.

> Compares the operands. Returns TRUE if the operand on the left is greater than the operand on the right, FALSE otherwise. If the operands are strings, then the ASCII values are compared; if they are numbers then the actual numeric values are compared.

< Compares the operands. Returns TRUE if the operand on the left is less than the operand on the right, FALSE otherwise. If the operands are strings, then the ASCII values are compared; if they are numbers then the actual numeric values are compared.

> = Compares the operands. Returns TRUE if the operand on the left is greater than or equal to the operand on the right, FALSE otherwise. If the operands are strings, then the ASCII values are compared; if they are numbers then the actual numeric values are compared.

< = Compares the operands. Returns TRUE if the operand on the left is less than or equal to the operand on the right, FALSE otherwise. If the operands are strings, then the ASCII values are compared; if they are numbers then, the actual numeric values are compared.

&& Logical AND. Returns TRUE if both operands have a value other than 0. Only integer operands are allowed.

|| Logical OR. Returns TRUE if either operand has a value other than 0. Only integer operands are allowed.

+ Adds integers or concatenates strings. If the operands are both integers, then the result is the sum of the integers. If the operands are both strings, then the result is the concatenation of the two strings.

- Subtracts integers. Only integer operands are allowed.

***** Multiplies integers. Only integer operands are allowed.

/ Divides integers. Only integer operands are allowed.

() Parentheses. Used to force an evaluation order. If parentheses are not used, the standard C language order of evaluation is used.

! Logical NOT. This is a unary operator. It only takes one operand. If the operand is 0, the result is TRUE. If the operand is nonzero, the result is FALSE.

In addition to these operators, there are constants that have special meaning. These constants check for conditions relating to the FILE entry that the expression appears in. These constants are:

ALWAYS
Always evaluates to TRUE. It can be used in the WHEN and EXITWHEN keywords.

DELETE
Evaluates to TRUE if a DELETE is being performed. Otherwise, evaluates to FALSE. It can be used in the WHEN and EXITWHEN keywords.

EXIST
Can be used only in the WHEN keyword. Evaluates to TRUE only if the file already exists on the workstation. You can use this value for files that are optional. If the user deletes the file after installation, it is not replaced when the product is updated.

FALSE
Evaluates to numeric 0. It can be used in the WHEN and EXITWHEN keywords.

INSTALL
Evaluates to TRUE if an installation is being performed. Otherwise, evaluates to FALSE. It can be used in the WHEN and EXITWHEN keywords.

NEVER
Always evaluates to FALSE. It can be used in the WHEN and EXITWHEN keywords.

NOTEXIST
Can be used only in the WHEN keyword. Evaluates to TRUE only if the file does not already exist on the workstation.

OUTOFDATE
Can be used only in the WHEN keyword. Evaluates to TRUE if the date and time of the existing workstation file are older than the date and time specified in the DATE and TIME keywords. This constant also evaluates to TRUE if the workstation file does not already exist on the workstation. In all other cases, it evaluates to FALSE.

RESTORE
Evaluates to TRUE if a RESTORE is being performed. Otherwise, evaluates to FALSE. It can be used in the WHEN and EXITWHEN keywords.

TRUE
Evaluates to numeric 1. It can be used in the WHEN and EXITWHEN keywords.

UPDATE
Evaluates to TRUE if an UPDATE is being performed. Otherwise, evaluates to FALSE. It can be used in the WHEN and EXITWHEN keywords.

Chapter 7. Description File

,,,,,,,

Autolink to Sibling Window

RES='6100' VPX='25%' VPY='bottom' VPCX='75%' VPCY='100%

7.1 Overview of the Description File

The description file is a text file that contains a brief description of the product. The installation and maintenance utility transfers this file to the workstation and displays the description when a user selects **Product description** from the Details pull-down menu.

The installation and maintenance utility displays the text in the description file exactly as you type it. The only formatting that occurs is word wrapping.

7.2 Creating a Description File

Use your text editor to create a description file. Comment lines are not permitted in a description file.

When creating the description file, keep in mind that the product description is displayed in the system's proportional font on the Product description window. This means that the lines on your Product description window might wrap causing the ends of the lines to not line up because characters can occupy various amounts of space in different fonts. You should experiment with where you end lines in your product description file until you have a description that is displayed in an attractive format.

Set the **PKGDESCRFILE** [**Link to heading 5.17 on page 23**] keyword in the PACKAGE entry of the catalog file to the file name that you choose for the description file.

7.3 Example of a Description File

Note: No comment lines are allowed in a description file.

The following text is in the sample description file for the fictitious SuperMarket Systems Software Inventory Control System.

```
The SuperMarket Systems Software Inventory Control System provides
automated inventory services for supermarket stock for the most
elegant food store chain.
```

Chapter 8. Customizing Your Product's Installation

,,,,,,,

Autolink to Sibling Window

RES='&hres.' VPX='25%' VPY='bottom' VPCX='75%' VPCY='100%

8.1 Tailoring the Initial Installation

The initial installation program (EPFIDLDS.EXE) displays an animated sequence of bit maps and starts the installation and maintenance utility (EPFINSTS.EXE) which uses your product's catalog and package files.

You can customize this installation process by modifying the resource file (IIRC.RC) provided with Software Installer.

This file contains string tables that define all of the tailorable attributes of the initial installation program. String tables consist of one or more string identifiers followed by a string literal. The string literal can be comprised of one or more parameters.

For example:

```
STRINGTABLE
   BEGIN
   /*------------------------------------------------------------*/
   /*   -------------- <          string literal              > */
   /*   ---------------- <parameter1> <parameter2> <parametern> */
   /*   ------------------------------------------------------------*/
   string1_identifier, " parm1_value, parm2_value, parmn_value"
   string2_identifier, " parm1_value, parm2_value, parmn_value"
   stringn_identifier, " parm1_value, parm2_value, parmn_value"
```

After you modify the IIRC.RC file, rebind it to EPFIDLDS.EXE using the BLDRC.CMD command file. BLDRC.CMD uses the Resource Compiler provided by the OS/2 2.0 Toolkit.

Any errors in your modification of the strings will be displayed by the installation utility.

When the compilation of the IIRC.RC and EPFIDLDS.EXE completes successfully, you will receive the following message.

Build completed successfully.
You can now run xxxIDLDS.EXE to test your changes.

Where *xxx* is the prefix you used when you renamed the installation and maintenance utility files.

8.2 Specifying the Sources and Host File Qualifiers

Use the strings identified by the following string identifiers in the IIRC.RC resource file to specify which sources (MVS, VM, VSE, or drive) are allowed and to specify the base (non-translatable) and NLS (translatable) qualifier for product files on those systems.

INST_SOURCES
Specifies which sources will appear on the installation source window. Any combination of MVS, VM, VSE, or DRIVE can be used. If more than one source is specified, a window asking the user to chose a source is displayed.

MVS_BASE_3Q
The third qualifier in the name of the source MVS data set that base installation parts are transferred from.

Note: MVS file names consist of four parts, such as *QUAL1.QUAL2.QUAL3(FILENAME)*. The first and second qualifiers, QUAL1 and QUAL2, are entered by the user in the MVS source dialog. The third qualifier is specified by MVS_BASE_3Q. The NLS and base parts of Software Installer can be separated into different data sets by specifying a different third qualifier for each set of parts.

See the following table for base parts and NLS part file names.

Base parts (non-translatable)	NLS parts (translatable)
---	---
EPFIEXTS.DLL	EPFIHPLB.HLP
EPFINSTS.EXE	EPFIMSG.MSG
EPFIPRCS.EXE	EPFIPII.DLL
EPFIUPCK.EXE	

MVS_NLS_3Q
The third qualifier in the name of the source MVS data set that NLS installation parts are transferred from.

VM_BASE_TYPE
The file type of the VM source files that base parts are transferred from.

VM_NLS_TYPE
The file type of the VM source files that NLS parts are transferred from.

VSE_BASE_Q
The file qualifier of the VSE source files that base parts are transferred from.

VSE_NLS_Q
The file qualifier of the VSE source files that NLS (translatable) parts are transferred from.

The system sources and qualifiers string table is as follows when you receive Software Installer.

```
STRINGTABLE
    BEGIN
        INST_SOURCES,     "MVS, VM, "VSE", DRIVE"
        MVS_BASE_3Q,      "SEPFBASE"
        MVS_NLS_3Q,       "SEPFBENU"
        VM_BASE_TYPE,     "BASEPF11"
        VM_NLS_TYPE,      "ENUEPF11"
        VSE_BASE_Q,       "BASEPF11"
        VSE_NLS_Q,        "ENUEPF11"
    END
```

8.3 Customizing the Initial Installation Window

The initial installation window is the window that all other installation windows appear on top of and that can optionally display bit maps.

The following strings are required:

EXIT_BUTTON_COORDS
The initial position and size of the Exit push button.

INST_WINDOW_SIZE
The initial size of the installation window. The window will be centered on the Workplace Shell.

INST_WINDOW_TITLETEXT
The title to appear on the title bar of the window.

INST_START_PD_TEXT
The text displayed as the first choice on the File pull down menu.

INST_BACKGROUND_COLOR
The color of the background of the window. Options are:

- BACKGROUND (same as the background of the desktop)
- BLACK
- BLUE
- BROWN
- CYAN
- DARKBLUE
- DARKCYAN
- DARKGRAY
- DARKGREEN
- DARKPINK
- DARKRED
- GREEN
- NEUTRAL
- PALEGRAY
- PINK
- RED
- WHITE
- YELLOW

To customize the initial installation window, modify the parameters in the following string table in IIRC.RC.

```
STRINGTABLE
   BEGIN
      /* ------------------------------------------------------- */
      /*                              x       y      width      height */
      /*                            ---     ---     -----      ------ */
      EXIT_BUTTON_COORDS,          "542,    325,       75,        35"
      INST_WINDOW_SIZE,            "630,    470"
      INST_WINDOW_TITLETEXT,       "Initial Installation"
      INST_START_PD_TEXT,          "~Start install..."
      INST_BACKGROUND_COLOR,       "darkblue"
   END
```

8.4 Loading Bit Maps in the Initial Installation Window

The initial installation window can display and horizontally move up to 100 individual bit maps. Define all of the bit maps you want to display in the window with a BITMAP line.

For example, if you are using three bit maps named EPFII1BS.BMP, EPFII2BS.BMP, and EPFII3BS.BMP, you should define them in the IIRC.RC file as follows:

```
BITMAP   INST_BITMAP1      EPFII1BS.BMP
BITMAP   INST_BITMAP2      EPFII2BS.BMP
BITMAP   INST_BITMAP3      EPFII3BS.BMP
```

Note that the bit map identifiers (INST_BITMAP1 through INST_BITMAP3) must be consecutive.

8.5 Customizing the Bit Maps

The bit map positions and movement characteristics in the initial installation window can be customized by modifying parameters in a string table in IIRC.RC.

The parameters of these strings represent coordinates in the window. All coordinates are relative to the upper-left hand corner of the window. A bit map's position coordinate should not be specified as negative or greater than the bounds of the window.

The bit maps are displayed one at a time starting with INST_BITMAP1_COORDS.

You can specify movement of a bit map from its initial position to its ending position in horizontal segments, or slices. You specify this by setting the *slices* parameter to a number greater than one. When this is done, the initial installation program divides the specified bit map into equal size slices and moves each one individually as specified by the movement parameters. Dividing a bit map into equal sized slices and moving them one at a time from a position off the screen to the final position is an effective way of moving large images onto the screen without the movement being jerky.

A delay can also be specified between the display of each bit map. You can use the *delay* parameter to add dramatic pauses and create animation in displaying your product's logos.

Use the following two parameters to specify the initial location of the bit map.

initx (required) The initial horizontal position of the bit map

y (required) The initial vertical position of the bit map

Use the following four parameters to move a bit map left or right.

endx (optional) The horizontal position that you want the bit map moved to. If endx is greater than initx, then the bit map is moved to the right the number indicated by mvinc until endx is reached. If endx is less than initx, then the bit map is moved to the left until endx is reached. If endx and initx are the same or endx is omitted, the bit map is not moved.

mvinc (optional) The number of picture elements (pels) to move the bit map at one time. This parameter determines the speed that the bit map moves from initx to endx. It must always be positive. If mvinc is omitted and endx is different than initx, an increment of 15 is used.

slices (optional) The number of horizontal segments to divide the bit map into. If the number of slices is greater than one, each slice of the bit map is drawn on the screen (one at a time) according to the previously specified parameters. The movement of the bit map starts with the top slice.

delay (optional) The amount of time (in tenths of a second) to wait before drawing the next bit map. If this parameter is omitted, then the next bit map is drawn immediately.

An example of the string table you need to modify follows.

```
STRINGTABLE
   BEGIN
      /* ------------------------------------------------------ */
      /*                      initx   y    endx  mvinc slices delay */
      /*                      -----  ----  ----  ----- ------ ----- */
      INST_BITMAP1_COORDS, " 400,  20 ,  200,   60,   20,      5 "
      INST_BITMAP2_COORDS, "  -1,  90,   200,  100,  100,     10 "
      INST_BITMAP3_COORDS, " 230, 170,   230,   60,  400,     30 "
   END
```

8.6 Customizing the Initial Installation Information Windows

The initial installation program displays two information windows, one immediately after the program is called and the other after the installation and maintenance utility is installed.

You change the following strings in a string table of the IIRC.RC file to customize these windows.

INFOn_DISPLAY
Flag that indicates whether to display the n information window, where n is either 1 or 2 for windows one and two, respectively. Valid values are:

1 Display the window.

0 Do not display the window. All other strings in this table are ignored.

INFOn_TITLE
The title that appears on the window.

INFOn_COORDS
The initial position and size of the window.

INFOn_FILE
The file containing the information to be displayed. This file name should be completely unqualified. The file will be searched for in the same directory that the program is running from. If this file is not found or if this string is omitted, the text specified by the parameters of INFOn_TEXTxx strings are used.

INFOn_TEXTxx
The lines of text to display in the window where xx is 1 - 10. The text on these lines is concatenated and word wrapped. Use \015 to specify that a new line is desired.

As example of the string table you should modify to change the information windows follows.

```
STRINGTABLE
   BEGIN
      /*************************************************************/
      /* Information window 1 - displayed after initial bit map.   */
      /*************************************************************/
      INFO1_DISPLAY, "1"
      INFO1_TITLE,   "Information Window 1"
      INFO1_COORDS,  "10, 100, 150, 100"
      INFO1_FILE,    "README.ME"
      INFO1_TEXT1,   "This is the first line of information.  Use "
      INFO1_TEXT2,   "\015 if you desire a new line.  Otherwise the "
      INFO1_TEXT3,   "lines will be concatenated together."
      INFO1_TEXT4,   "\015\015This is info line 4."

      /*************************************************************/
      /* Information window 2 - displayed after installation of    */
      /* the installation utility.                                 */
      /*************************************************************/
      INFO2_DISPLAY, "1"
      INFO2_TITLE,   "Information Window 2"
      INFO2_COORDS,  "10, 100, 150, 100"
      INFO2_TEXT1,   "This is the first line of information.  Use"
      INFO2_TEXT2,   "\015 if you desire a new line.  Otherwise the"
      INFO2_TEXT3,   "lines will be concatenated together."
      INFO2_TEXT4,   "\015\015 This is info line 4."
   END
```

8.7 Customizing the Prompting Windows

Depending on the value of the *INST_SOURCES* string, several windows are displayed that prompt for installation information.

Use the following strings to customize defaults for installation information.

INST_DESTINATION
The default destination directory that will appear on all installation windows (MVS, VM, VSE, and DRIVE). Be sure to use double backslashes (\\) to represent a backslash.

SOURCE_DIALOG_POS
The initial position of the installation source window.

DRIVE_DIALOG_POS
The initial position of the Install from drive window.

MVS_DIALOG_POS
The initial position of the Install from MVS host window.

MVS_SOURCE_DSN
The name of the default MVS source data set that appears on the Install from MVS host window.

VM_DIALOG_POS
The initial position of the Install from VM host window.

VM_SOURCE_MINIDISK
The default VM minidisk that appears on the Install from VM host window. This must be a single character. An '*' causes the CMS search order to be used.

VSE_DIALOG_POS
The initial position of the Install from VSE host window.

VSE_SOURCE_NAME
The name of the default VSE source that appears on the Install from VSE host window.

PROGRESS_DIALOG_POS
The position of the initial installation Progress window.

The string table you should modify to change the prompting window follows.

```
STRINGTABLE
    BEGIN
        INST_DESTINATION,       "C:\\MYINSTALL"
        SOURCE_DIALOG_POS,      "0,   100"
        DRIVE_DIALOG_POS,       "0,   100"
        MVS_DIALOG_POS,         "0,   100"
        MVS_SOURCE_DSN,         "EPF.V1R2M0"
        VM_DIALOG_POS,          "0,    100"
        VM_SOURCE_MINIDISK,     "*"
        VSE_DIALOG_POS,         "5,   5"
        VSE_SOURCE_NAME,        "EPF.V1R2M0"
        PROGRESS_DIALOG_POS,    "0,    100"
    END
```

8.8 Customizing the Startup Parameters

The installation and maintenance utility must be called from the user's workstation to actually perform the product installation. However, you can specify startup parameters that are passed to the installation and maintenance utility by modifying the following strings in IIRC.RC.

STARTPARM_CATALOG_DRIVE
The unqualified catalog file name used by the installation utility if the installation source is a drive.

STARTPARM_CATALOG_MVS
The unqualified catalog file name used by the installation utility if the installation source is MVS. (member name only - the data set name is determined by what the user enters and the value of MVS_NLS_3Q).

STARTPARM_CATALOG_VM
The unqualified catalog file name used by the installation utility if the installation source is VM. (file name only - the type is specified by VM_NLS_TYPE).

STARTPARM_CATALOG_VSE
The unqualified catalog file name used by the installation utility if the installation source is VSE. (file name only - the qualifier is specified by VSE_NLS_Q).

STARTPARM_PRODUCT
The name of the product being installed as it appears in the NAME keyword of the PACKAGE entry in the catalog file.

STARTPARM_ACTION
The action to be performed by the installation utility (i=install, u=update, d=delete, or r=restore).

STARTPARM_POSITION
The initial position of the installation utility.

STARTPARM_RESPFILE
The workstation name of the **response** [**Link to heading 16.4 on page 113**] file that the installation utility is called by. This string can be omitted if you do not use a response file.

The string table you should modify to change the startup parameters follows.

```
STRINGTABLE
    BEGIN
        STARTPARM_CATALOG_DRIVE, "epficat.icf"
        STARTPARM_CATALOG_MVS,   "epficat"
        STARTPARM_CATALOG_VM,    "epficat"
        STARTPARM_CATALOG_VSE,   "epficat"
        STARTPARM_PRODUCT,       "Sample Product"
        STARTPARM_ACTION,        "i"
        STARTPARM_POSITION,      "0,  100"
    END
```

Chapter 9. Registering Installation Classes and Creating Objects for the Workplace Shell

Software Installer provides you the capability to register a workplace class and create workplace objects for your product as part of the installation process. This creation of product classes in the user's Workplace Shell* is transparent to the end user.

To register product classes, use the EXIT keyword of a FILE entry in the package file to call the **REGISTERWPSCLASS**[**Link to heading 13.22 on page 98**] installation exit.

The syntax of this exit is:

```
FILE
  EXIT = 'REGISTERWPSCLASS classname dllname'
```

Register your classes before creating instances (workplace objects) of them.

After you have registered your product classes, you should create workplace objects (instances of these classes), using the **CREATEWPSOBJECT**[**Link to heading 13.14 on page 93**] installation exit.

The syntax of this exit is:

```
FILE
  EXIT = 'CREATEWPSOBJECT classname title location replace_flag setupstring'
```

Chapter 10. Compressing Your Source Files

,,,,,,,

Autolink to Sibling Window

RES='8100' VPX='25%' VPY='bottom' VPCX='75%' VPCY='100%

- **File compression** [**Link to heading 10.1 on page 80**]
- **EPFIPACK syntax and parameters** [**Link to heading 10.2 on page 80**]

10.1 File Compression

Compressing your source files before you load them onto the shipping media saves space. You can use your own compression utility or you can use the compression service provided by the renamed EPFIPACK utility.

If you use the **Diskette Generator Utility**[**Link to heading Chapter 4 on page 14**], file compression is done automatically.

You can make the packed file name anything you want. However, EPFIPACK packs the source file using the name as it was when you issued the pack command. In your .PKG file, your FILE entry SOURCE keyword contains the name of the packed file and the FILE entry PWS keyword must contain the name of the file that was packed.

Two types of file compression are supported by Software Installer.

Individual file compression
Each product file is packed separately into a smaller file. If a product contains 10 files and uses individual file compression, 10 compressed files are created (one from each product file).

When the installation and maintenance utility installs a product that uses individual file compression, it unpacks each file after transferring it to the workstation.

You specify individual file compression by using the UNPACK keyword of the **FILE**[**Link to heading 6.25 on page 48**] entry in the package file.

Multiple file compression
Multiple product files are packed into a single packed file.

When the installation and maintenance utility installs a product that uses multiple file compression, it transfers and unpacks the packed file into individual files.

You specify multiple file compression by using the PACKFILE entry type and the PACKID keyword of the FILE entry in the package file.

When either type of file compression is used, the compressed file is deleted from the user's workstation after unpacking.

10.2 EPFIPACK Syntax and Parameters

```
EPFIPACK [d:][path]source_filename[.ext]
         [d:][path][packed_filename[.ext]] [/L]
```

Parameters

source_filename
If /L is not specified, this is the file to be compressed. If /L is specified, this file is not compressed but instead is interpreted as a **list_filename**[**Link to footnote, ID=lstfnam on page 81**].

The following rules apply to the source_filename.

- The source_filename is required when using the EPFIPACK command.
- The file name must be a valid OS/2 file specification.

packed_filename
This is the destination file that will contain the compressed data. This must be a valid OS/2 file specification. If /L is not specified and the source parameter consists of only a path or has a global file name character specified in source_filename, this file will contain the compressed data for the multiple source files. If /L is specified, this file will contain the compressed data for all of the files specified in the list_filename file. If neither the packed_filename nor /L is specified, the destination file name is constructed based on the source_filename. The destination file name is the source_filename modified to contain an @ as the last character of the file name extension. If the packed_filename is not specified but /L is, the list_filename will be used as the destination file name and modified to contain an @ as the last character of the file name extension. The file contents will be the end-to-end compressed data for the files listed in the list_filename file.

/L
EPFIPACK parameter that specifies that the source parameter should be interpreted as list_filename instead of source_filename. This means that the list_filename file should not be compressed because it actually contains the list of files that are to be compressed.

Table 2. Footnote Panels for Current Heading

Footnote ID	Footnote
lstfnam	list_filename When /L is specified with EPFIPACK, source_filename is interpreted to be the name of a file containing a *list* of files to compress. The list file itself should not be compressed. The format of the data in the list file is: [d:][path]source_filename[.ext] The following rules apply to the list file. • The packed_filename cannot be included in the list file because it was specified on the command line. • Global characters are not permitted. • Nested list_filenames are not permitted. • A separate line must be used for each file to be compressed. • You can add a comment or a blank line inside the list_filename file by having a semicolon (;) in the *first* column of that line.

Chapter 11. Updating Files That Are in Use

When updating a component, the possibility exists that a file the component uses is in use. For example, Workplace Shell* loads DLL files for any registered classes. Because Workplace Shell is using the DLL files for these classes, some of the updated DLL files cannot be replaced while Workplace Shell is active. You can overcome this conflict by using the **FILE entry** [**Link to heading 6.25 on page 48**] keyword REPLACEINUSE.

The keyword REPLACEINUSE accepts any combination of four character values to indicate when the user wants the in-use copy of a file to be acted upon. The characters are:

'D' Replace on delete processing
'I' Replace on install processing
'R' Replace on restore processing
'U' Replace on update processing

To specify multiple values, enclose all values in single quotes. For instance:

REPLACEINUSE = 'I U R'

would replace the in-use copy of a file for installation, updating, and restore processing.

When the installation and maintenance utility finds that the REPLACEINUSE keyword is specified for a file and that the file is in use, the file is copied to a temporary file. If there is not enough space in the temporary location, the user receives a warning message and the installation processing stops.

The action on the in-use file is deferred until the next start of the operating system. The installation and maintenance utility does this by temporarily replacing Workplace Shell* with its own shell. This allows most files to be replaced before the OS/2 system loads them when it is restarted.

Important: When the requested action is complete, the user receives a message to shutdown the system and restart the operating system to activate the changes. You can use an **OPTIONS entry**[**Link to heading 6.35 on page 56**] in the package file to change the text of this message.

This does not work for files loaded by CONFIG.SYS commands because these files are loaded before the shell is started. To replace these files, instruct the user to replace them manually.

Chapter 12. Using Hooks to Change Installation Directories

You can customize the EPFIHOOK.C program to perform special directory processing your product needs when it is installed.

Software Installer calls your renamed **hook** [**Link to heading Chapter 24 on page 127**] function each time a directory entry field is changed by the user on the Install - directories window.

To implement this customization, you should rebuild the renamed EPFIHOOK.DLL after **running ISREN.CMD**[**Link to heading Chapter 15 on page 105**].

1. Change the LIBRARY keyword name in your renamed EPFIHOOK.DEF from EPFIHOOK to xxxIHOOK where xxx is your product prefix.
2. Change xxxIHOOK.C to perform the processing you want.

   The original EPFIHOOK.C contains code that will apply changes to all Install - directories when the user changes any directory entry field. All directories are essentially grouped into one root.
3. Change every occurrence of EPF to xxx in xxxIHOOK.MAK.
4. Rebuild the xxxIHOOK.DLL by running NMAKE as follows.

   NMAKE /F xxxIHOOK.MAK
5. Test the results of your hook by starting your product's installation and checking for the expected results of your changes to xxxIHOOK.C.
6. Run BLDINST.CMD. INSTALL.IN_ packages your hook so that it will be in the same directory as xxxINSTS.EXE.

Chapter 13. Exiting During Installation Processing

,,,,,,,

Autolink to Sibling Window

RES='10100' VPX='25%' VPY='bottom' VPCX='75%' VPCY='100%

- **Using exits to perform product-specific tasks** [**Link to heading 13.1 on page 84**]
- **Creating installation exits** [**Link to heading 13.2 on page 85**]
- **Calling installation exits** [**Link to heading 13.3 on page 85**]
- **Substituting Installation Variables** [**Link to heading 13.4 on page 86**]
- **Passing data between Software Installer and exits** [**Link to heading 13.5 on page 86**]
- **Default installation exits** [**Link to heading 13.9 on page 89**]

13.1 Using Exits to Perform Product-Specific Tasks

Installation exits allow you to perform product-specific tasks by running DLLs, EXEs, or command files outside of Software Installer during install, update, restore, or delete processing.

For instance, you may want to run an exit that checks for application-specific requirement during one of the actions when a Progress window is displayed. You may want to stop processing without returning message *EPFIE187: A product-specific installation error occurred while executing the '%2' exit routine. The return code is %1.*.

You can do this by using the **getowner()** [**Link to heading 13.6 on page 87**] and WinPostMsg() functions in your exit.

```
#include <varsfp.h>
HWND   hWndProgress;

hWndProgress=getowner();
WinPostMsg(
   hWndProgress,
   WM_COMMAND,
   (MPARAM) INST_PROGRESS_STOP_BUTTON,
   (MPARAM) NULL);
```

This stops the processing by simulating pressing the **Stop** push button on the Progress window. The user will still get a message that the action in progress was cancelled. Your exit should prepare the user for this message by telling the user to restart the requested action when notified that it has been cancelled.

Software Installer also provides several **installation exits** [**Link to heading 13.9 on page 89**] that you can use to perform useful installation tasks.

13.2 Creating Installation Exits

You can create your exits as either DLL entry points or as a EXE or CMD executable. To run an external DLL, EXE, or CMD file, modify your package file to use the **EXIT entry**[**Link to heading 6.20 on page 47**].

DLL entry point exits

1. Create an exit DLL file with one or more entry points in it.

   Each of these entry points can perform a different product-specific installation task

   Each of the entry points in the DLL file should be declared as follows:

   USHORT EXPENTRY FunctionName(CHAR *pzParameterString);

   Declare the function as USHORT so that the installation and maintenance utility can display a message to the user when a nonzero return code is returned.

   pzParameterString
   The parameter string that appears after the entry point name when the exit is called using the EXIT keyword in a FILE entry of the package file. Once the function is started, the installation exit can use the parameter string in any manner.

2. Modify your package file to use the **EXIT entry** [**Link to heading 6.20 on page 47**] to specify the DLL file that contains your installation exits.
3. Add an EXIT keyword to the FILE entry **to call the specified installation exit**[**Link to heading 13.3 on page 85**].

EXE or CMD file exits

1. Create your exit as an EXE or CMD file.

   This method might be slower than a DLL exit, but it is useful for running existing programs.

2. Use the **EXEC**[**Link to heading 13.21 on page 97**] exit provided by Software Installer to run the EXE or CMD file as an exit.

13.3 Calling Installation Exits

Use the EXIT keyword in the FILE entry to actually start an installation exit. This keyword specifies the function in the DLL file that you specified with the EXIT entry. If you omit it from a FILE entry, no installation exit is started for that file.

The EXIT keyword value is a text string with a maximum length of 1024 bytes. If it contains spaces, commas, or single quotes, you must enclose it in single quotes. The installation and maintenance utility interprets the text string in two parts: an entry point and a parameter string. These parts are separated by one or more spaces.

You might find it useful to use **variable substitution** [**Link to heading 13.4 on page 86**] for the keyword value.

The following FILE entry shows the format of the EXIT keyword.

```
FILE
  EXIT = 'FUNCTION parameter-string'
```

The DLL file specified in the EXIT entry must be transferred to the workstation before an installation exit in the DLL file is started. This means that the FILE entry that transfers the DLL file should occur before any EXIT keywords are specified. An installation exit is not called if the file you specify in the FILE entry is not transferred. A message is displayed for this error.

When writing package files that start installation exits, you can assume that the current directory is the FILE directory specified in the Install - directories window. You *cannot* assume that the current directory is the directory that the last file was transferred into.

13.4 Substituting Installation Variables

You can use installation variable substitution for many keyword values in the package file. Using installation variable substitution makes it easy to reference any **installation variable** [**Link to heading Chapter 17 on page 118**] from the package file.

Note: Defined variables are not case-sensitive.

Any installation variable enclosed by % symbols is replaced with the value of that variable.

For instance, if the variable **VAR** is set to a value of 'TEXT.EXE', the package file FILE entry EXIT keyword is written as follows:

```
EXIT = 'EXEC fg tw %VAR%'
```

After substitution, the value is:

```
EXIT = 'EXEC fg tw TEST.EXE'
```

You can use variable substitution for the value of the following keywords:

FILE entry type[**Link to heading 6.25 on page 48**]:

- EXIT
- EXITWHEN
- PWS
- UNPACK
- WHEN

EXIT entry type[**Link to heading 6.20 on page 47**]:

- DLL

PACKFILE entry type[**Link to heading 6.40 on page 58**]:

- EXIT
- EXITWHEN
- UNPACK

If your installation specifies HPFS files or directories, enclose the file or directory variables with double quotes. The double quotes are necessary to preserve any spaces in the HPFS names.

13.5 Passing Data between Software Installer and Exits

Software Installer provides the following APIs for passing information between itself and installation exits. You can use these APIs in installation exits written in C.

getowner [**Link to heading 13.6 on page 87**]
Retrieves the handle of the installation Progress window

getvar [**Link to heading 13.7 on page 88**]
Retrieves the value of an installation variable

putvar [**Link to heading 13.8 on page 89**]
Sets the value of an installation variable referenced in a package file

To use these APIs, your installation exit code must include VARSFP.H and you must link to VARS.OBJ when you compile the exit. These files are installed with Software Installer.

Note: You must use a 32-bit compiler, such as the IBM C-Set/2 compiler, when compiling your exits that use these functions.

13.6 getowner API

You can use getowner to retrieve the window handle of the installation Progress window. You might want to do this in your installation exits when windows started in your exit require a user to enter information before continuing to work in the installation Progress window.

Syntax

```
#include <varsfp.h>

HWND hwndOwner;

hwndOwner =(hwnd)getowner();
```

Example

This sample program uses the getowner() function call to query the handle of the installation Progress window. It uses this handle as the owner of the secondary window.

```
int main(void)
   {
   HWND   hOwner;
   HAB    hAB;
   HMQ    hmq;

   hAB = WinInitialize((ULONG) NULL);
   hmq = WinCreateMsgQueue(hAB, 0);

   /************************************************************/
   /* Get the handle of the Progress window so that the message */
   /* box can be created with the correct owner.                */
   /************************************************************/
   hOwner = getowner();
   if (!hOwner)
      hOwner = HWND_DESKTOP;

   /************************************************************/
   /* Display a secondary window with sample text.  Notice that */
   /* the owner is the window handle queried using the getowner() */
   /* function call.                                            */
   /************************************************************/
   WinMessageBox(
      HWND_DESKTOP,
      hOwner,
      "This is the text that is displayed in the secondary window",
      "Secondary window title",
      (USHORT) 0,
      (ULONG) (MB_APPLMODAL | MB_MOVEABLE | MB_OK));

   return(0);
   }
```

13.7 getvar API

Use getvar to retrieve the value of a specified installation variable.

A program that has been called during the installation processing by an EXIT entry type in the package file can use the getvar API to obtain data which has been set by the installation and maintenance utility or another installation exit.

Syntax

```
#include <varsfp.h>

ULONG    ulRC;
CHAR *   pszVarName;
CHAR     szBuffer[BUFFSIZE];
ULONG    ulBufferSize;

ulBufferSize =BUFFSIZE;

ulRC =getvar(pszvarname, szBuffer, &ulBufferSize);
```

Parameters

ulRC
The return value. Valid values are:

1 Successful function.

0 Error occurred. Possible causes include insufficient memory, no variable specified, variable not found.

pszVarName — required
A null-terminated string that contains the name of the installation variable to be retrieved. This name is not case-sensitive.

szBuffer — required
Buffer where the installation and maintenance utility returns the value of the specified installation variable.

&ulBufferSize — required
Pointer to the size of the buffer. The actual number of bytes specified by *ulBuffersize* is copied to the buffer. The number of bytes copied is returned in this variable.

13.8 putvar API

Use putvar to set the value of a specified installation variable.

By using putvar, an exit can return data to the installation and maintenance utility that can be used by your installation processing.

Syntax

```
#include <varsfp.h>

ULONG   ulRC;
CHAR    *pszVarName, *pszVarValue;

ulRC =putvar(pszVarName, pszVarValue);
```

Parameters

ulRC
The return value. Valid values are:

1 Successful function.

0 Error occurred. Possible causes include insufficient memory, no variable specified, variable not found.

pszVarName — required
A null-terminated string containing the name of the installation variable to be set. This name is not case-sensitive.

pszVarValue — required
A null-terminated string containing the value of the installation variable to be set. Only character data is supported. If this parameter is set to null, the installation variable is deleted.

13.9 Default Installation Exits

The installation and maintenance utility is shipped with a DLL file named EPFIEXTS.DLL. This file contains installation exits DLLs to accomplish the following common installation tasks.

13.10 ADDCONFIG

Use the ADDCONFIG installation exit to append parameters to the CONFIG.SYS file.

ADDCONFIG only appends to the CONFIG.SYS file if the user selected the **Update CONFIG.SYS** check box in the Install window or the Update window.

Installation variable substitution [**Link to heading 13.4 on page 86**] is useful in this exit.

The syntax of this exit is:

FILE
EXIT = 'ADDCONFIG parameter'

parameter
The line that you are adding at the end of the CONFIG.SYS file.

Example

The following FILE entry shows how you can base a new installation variable on an existing installation variable using variable substitution.

```
FILE
  EXIT = 'ADDCONFIG SET new_variable=%existing_variable%'
```

Do not use this installation exit to update LIBPATH because the value of LIBPATH exists only in the CONFIG.SYS file. The UPDATECONFIG exit can be used to update LIBPATH.

13.11 ADDINI

Use the ADDINI installation exit to add OS2.INI application name/keyname pairs and their corresponding values to the **OS2.INI file**[**Link to footnote, ID=osini on page 91**]. This exit can also be used to delete keynames from the OS2.INI file. If the application name you specify does not exist, it is created. If the keyword already exists, it is overwritten.

The syntax of this exit is:

```
FILE
  EXIT = 'ADDINI "appname" "keyname" "keyvalue"'
```

appname
OS2.INI application name to add the keyname to. If the application name you specify does not exist, it is created.

keyname
OS2.INI keyname to add. If the keyname does not exist, it is created. If the keyname already exists, it is overwritten.

keyvalue
Value of the keyname to add. If you omit this parameter, the specified keyname is deleted.

Example

The following example adds a keyname to the OS2.INI configuration file.

```
FILE
  EXITIGNOREERR = YES,
  EXITWHEN = 'INSTALL || UPDATE',
  EXIT = 'ADDINI "Sample Application Name" "Sample keyname" "Sample value"'
```

Table 3. Footnote Panels for Current Heading

Footnote ID	Footnote
osini	For more information on the structure of the OS2.INI file, see the *IBM Operating System/2 Version 2.0 PM Programming Guide* and the *IBM Operating System/2 Version 2.0 Programming Reference*.

13.12 ADDPROFILE

Use the ADDPROFILE installation exit to add an application name/keyname pair and its corresponding values to the specified OS/2 application profile (.INI) file. This exit can also be used to delete keynames from the OS2.INI file.

The syntax of this exit is:

```
FILE
  EXIT = 'ADDPROFILE Profile-name "appname" "keyname" "keyvalue"'
```

Profile-name
Name of the OS/2 application profile file to add the application name/keyname pair to. If the profile file does not exist, it is created.

appname
Application name to add the keyname to. If the application name does not exist, it is created.

keyname
Keyname to add. If the keyname does not exist, it is created. If the keyname already exists, it is overwritten.

keyvalue
Value of the keyname to add. If you omit this parameter, the specified keyname is deleted.

Example

The following example adds a keyname to the SAMPINI.INI file located in the file directory that the user specified as the location to install the product. The installation and maintenance utility sets the EPFIFILEDIR installation variable to the File directory specified on the Install - directories window.

```
FILE
  EXITIGNOREERR = YES,
  EXITWHEN      = 'INSTALL UPDATE',
  EXIT          = 'ADDPROFILE %EPFIFILEDIR%\SAMPINI.INI "Sample appname"
                              "Sample keyname" "Sample value"'
```

13.13 ADDPROG

Use the ADDPROG installation exit to add a program to the Workplace Shell.

The syntax of this exit is:

```
FILE
  EXIT = 'ADDPROG progtype "foldername" "progname"
         filename work-dir (parm1...parmN'
```

progtype
OS/2 execution type of the program.

There are two valid values:

PM The program uses Presentation Manager*.
OTHER The program does not use Presentation Manager.

foldername
Specifies the name of the Workplace Shell folder that the program is to be added to. If the name contains spaces, it must be enclosed in double quotes. If the folder does not already exist, it is created.

progname
Specifies the program title that appears in the Workplace Shell under the program's icon. If the name contains spaces, it must be enclosed in double quotes. If the program already exists, its properties (path, working directory, parameters, and program type) are overwritten by the new ones.

filename
Specifies the path and file name of the program to be added.

work-dir
Specifies the working directory for the program. This directory is the one changed to before executing the registered program.

parm1...parmN
Specifies command line parameters, if any, to be passed to the program whenever it runs. Separate these parameters by spaces.

The following example creates a folder named **Sample Folder** and adds a program called **Sample Program** in the **Sample Folder**. The Presentation Manager program that runs when an end user selects **Sample Program** from the Workplace Shell is D:\IBM\ADWP\SAMPLE.EXE. The working directory for the program is D:\IBM\ADWP.

```
PATH
  FILE = 'd:\ibm\adwp',

FILE
  SOURCE = 'MVS: PROJ.GROUP.TYPE(SAMPLE)';   VM: SAMPLE EXEBIN *;
   DRIVE: SAMPLE.EXE',
  PWS = SAMPLE.EXE,
  DATE = '921010',
  TIME = '1320',
  SIZE = 1320,
  EXITIGNOREERR = NO,
  EXITWHEN = 'INSTALL || UPDATE',
  EXIT = 'ADDPROG.PM "Sample Folder" "Sample Program" %EPHCURPWS% %EPHFILEDIR%'
```

13.14 CREATEWPSOBJECT

Use the CREATEWPSOBJECT exit to create a workplace object (instance) of an existing object class. This can be any one of a number of system provided objects, or an instance of the product provided classes.

Note: If you use OS/2 Version 2.0, you must have ServicePak XR06055 installed. If you do not have this ServicePak, all of your installation objects will default to the Software Installation icons.

The syntax of this exit is:

```
FILE
  EXIT = 'CREATEWPSOBJECT classname title location replace_flag setupstring'
```

classname - required
Object class of the object that is being created

title- required
Name of the object that is being created

location- required
Where the object is created (the folder name)

replace_flag- required
Whether the create should replace an object or fail if the object already exists. Valid values are:

F The create function will fail

R The create function will replace the existing object

setupstring - optional
Data that is sent to the object class at object creation time, and is object class dependent.

Note: There should be no blanks in the setupstring.

For more information about the setupstrings for OS/2 objects, see Chapter 9 of the *IBM Operating System/2 Version 2.0 PM Programming Reference Volume 2* (S10G-6265).

Example 1

The following FILE entry creates a folder to contain Market Tools! product objects.

```
FILE
  HOST = 'DRIVE:\MARKET.ICO',
  PWS = MARKET.ICO,
  WHEN = 'OUTOFDATE',
  EXITWHEN = 'INSTALL || UPDATE',
  EXIT = 'CREATEWPSOBJECT WPFolder "Market Tools!"
    <WP_DESKTOP> R "ICONFILE=%EPFIFILEDIR%\MARKET.ICO;OBJECTID=<MARKET1>"'
```

The following FILE entry creates a program reference object in the new folder.

```
FILE
  HOST = 'DRIVE:\MARKET.EXE',
  PWS = MARKET.EXE,
  WHEN = OUTOFDATE,
  EXITWHEN = 'INSTALL || UPDATE',
  EXIT = 'CREATEWPSOBJECT WPProgram "Speedy Delivery"
    <MARKET1> R "EXENAME=%EPFIFILEDIR%\MARKET.EXE;OBJECTID=<MEXE1>"'
```

Example 2

The following FILE entry creates an invoice object in the market folder.

```
FILE
  EXITWHEN = 'INSTALL || UPDATE',
  EXITIGNOREERR = YES,
  EXIT = 'CREATEWPSOBJECT INVOICE "INVOICE#1"
    <MARKET1> R OBJECTID=<INVOICE1>'
```

13.15 DELETEFILES

Use the DELETEFILES installation exit to delete data files created by your application.

The syntax of this exit is:

```
FILE
  EXIT == 'DELETEFILES file1...fileN'
```

file1...fileN
Name of one or more files you want to delete. File names can contain **global file name characters** [**Link to heading Chapter 23 on page 126**] (* and ?). If you use global file name characters, all files that match the specification are deleted.

The following example deletes all files that end with a .DAT extension and a .TMP extension from the work directory. This exit is only executed when the product is deleted (EXITWHEN='DELETE').

```
FILE
  EXITIGNOREERR = YES,
  EXITWHEN = ' DELETE' ,
  EXIT = ' DELETEFILES %EPFIWORKDIR%//*.dat %EPFIWORKDIR%//*.tmp'
```

13.16 DELETEINI

Use the DELETEINI installation exit to delete an application name from the **OS2.INI configuration file**[**Link to footnote, ID=osini on page 91**].

The syntax of this exit is:

```
FILE
  EXIT = ' DELETEINI appname1 appname2...appnamen'
```

appname1 appname2... appnamen
Are the names to delete. At least one application name must be specified. If any of the application names contain keynames and values, they are all deleted.

This exit is useful if your product needs to re-initialize a profile when a user updates the product. You can also use it to remove a profile when a user deletes the product.

13.17 DELETEPROFILE

Use the DELETEPROFILE installation exit to delete entire application names from a specified OS/2 application profile file.

The syntax of this exit is:

```
FILE
  EXIT = ' DELETEPROFILE Profile-name "appname1" "appname2"... "appnamen"'
```

Profile-name
Name of the OS/2 application profile to delete the application names from.

appname1 appname2... appnamen
Are the names to delete. At least one application name must be specified. If any of the application names contain keynames and values, they are all deleted.

This exit is useful if your product needs to re-initialize a profile when a user updates or deletes the product.

Example

The following example deletes two application names from the EPFINI.INI file located in the file directory:

```
FILE
  EXITIGNOREERR = YES,
  EXITWHEN = ' DELETE' ,
  EXIT = ' DELETEPROFILE %EPFIFILEDIR%//EPFINI.INI "App 1" "App 2"'
```

13.18 DELETEPROG

Use the DELETEPROG installation exit to delete programs from a specified folder in the Workplace Shell. When the last program is deleted from a folder, the folder is deleted.

The syntax of this exit is:

```
FILE
  EXIT = 'DELETEPROG "Foldername" "Progname"'
```

Foldername
Name of the Workplace Shell folder that the program is to be deleted from. This parameter is required and is case-sensitive. If the name contains spaces, it must be enclosed in double quotes.

Progname
Name of the program to be deleted. This parameter is required and is case-sensitive. If the name contains spaces, it must be enclosed in double quotes.

Example

The following example FILE entry shows an exit that deletes "Sample Program" from "Sample Folder" when the product is deleted:

```
FILE
  EXITIGNOREERR = YES,
  EXITWHEN = 'DELETE',
  EXIT = 'DELETEPROG "Sample Folder" "Sample Program"'
```

13.19 DELETEWPSOBJECT

Use the DELETEWPSOBJECT exit to delete a workplace object. This exit should be used before deregistering a class when deleting a product.

The syntax of this exit is:

```
FILE
  EXIT = 'DELETEWPSOBJECT <object_id>'
```

object_id
The OS/2 identifier of the object that is being deleted

Example

The following FILE entry deletes the INVOICE1 object:

```
FILE
  EXITWHEN = 'DELETE',
  EXITIGNOREERR = YES,
  EXIT = 'DELETEWPSOBJECT <INVOICE1>'
```

13.20 DEREGISTERWPSCLASS

Use the DEREGISTERWPSCLASS exit to deregister a workplace object class. Use this exit when deleting the product.

The syntax of this exit is:

```
FILE
  EXIT = 'DEREGISTERWPSCLASS classname'
```

classname
The name of the object class that is being deregistered

Example

The following deregisters the INVOICE class:

```
FILE
  EXITWHEN = 'DELETE',
  EXITIGNOREERR = YES,
  EXIT = 'DEREGISTERWPSCLASS INVOICE'
```

13.21 EXEC

Use the EXEC exit to start .CMD files or executable programs during installation. This installation exit calls whatever parameters follow it just as though they were entered at an OS/2 command prompt.

Note: If you use OS/2 Version 2.0, you must have ServicePak XR06055 installed. If you do not have this ServicePak, your installation will not return the appropriate results from your exit even though the exit completes correctly.

The syntax of this exit is:

```
FILE
  EXIT = 'EXEC where_executed sessiontype program-name parm1...parmN'
```

where_executed - optional
Specifies whether the called program runs in foreground or background. Valid values are:

FG (default) The program runs in the foreground.
BG The program runs in an invisible window in the background. Programs which run in the background should not prompt for input or display any output.

sessiontype - optional
Specifies the OS/2 session type. Valid values are:

PM Start the program as a Presentation Manager program.

FS Start the program as an OS/2 full-screen program.

TW Start the program as an OS/2 text-windowed program.

If you omit this parameter, the operating system chooses how to start the program.

program-name - required
File name of the program or command file to be executed.

Note: The program must contain an extension of either .CMD or .EXE and must exist in the FILE directory or in a directory referenced by the PATH installation variable, unless a fully qualified file name is given.

parm1...parmN
Command line parameters, if any, to be passed to the program.

Example

The following example calls the file that the installation and maintenance utility has just transferred.

```
FILE
  NAME = file1,
  SOURCE = ' MVS: TYPE(FILE1); VM: FILE1 ENUEPF11 *;
    DRIVE: FILE.exe' ;
  PWS = FILE1.EXE,
  SIZE = 12345,
  DATE = 920531,
  TIME = 1000,
  EXIT = ' EXEC fg tw %EPFICURPWS%'
```

Using the EXEC Installation Exit to Execute Command Processor Commands

You can use the EXEC installation exit to execute commands that exist in the CMD.EXE command processor. To do this, you must execute the CMD.EXE shell with a /c parameter. The following FILE entry shows how to use EXEC to execute a CMD.EXE copy command.

```
FILE
  NAME = ' Copy sample' ,
  DOWNLOAD = STATIC,
  WHEN = OUTOFDATE,
  SOURCE = ' MVS: SEPFBENU(EPFFILE); VM: EPFFILE ENUEPF12 *;
    DRIVE: EPFFILE.EXE' ,
  PWS = ' EPFFILE.EXE' ,
  DATE = 920101,
  TIME = 1200,
  EXIT = ' EXEC bg tw CMD.EXE /c COPY %epficurpws% %epfiworkdir%' ,
  EXITIGNOREERR = NO,
  EXITWHEN = ' INSTALL || UPDATE'
```

This file entry transfers the EPFFILE.EXE file and then copies it to the WORK directory that the user specified in the Install - directories window. This exit is executed in the background and is invisible to the user.

13.22 REGISTERWPSCLASS

Use the REGISTERWPSCLASS exit to register a workplace object class. The class is immediately available for creation of objects (instances) of the class's type after the exit completes. If the class already exists, an error is returned.

The syntax of this exit is:

```
FILE
  EXIT = ' REGISTERWPSCLASS classname dllname'
```

classname
The name of the object class that is being registered

dllname
The file name of the DLL file which contains the object class that is currently being registered.

Example

The following FILE entry registers the INVOICE class:

```
FILE
  HOST = 'DRIVE:\INVOICE.DLL',
  PWS = INVOICE.DLL,
  EXITWHEN = 'INSTALL || UPDATE',
  EXITIGNOREERR = NO,
  EXIT = 'REGISTERWPSCLASS INVOICE %EPFIFILEDIR%\INVOICE.DLL'
```

The %EPFIFILEDIR% substitution ensures that the class is registered in the directory that the INVOICE.DLL was installed into.

13.23 SETVAR

Use the SETVAR installation exit to set installation variables in the current installation execution.

The syntax of this exit is:

```
FILE
  EXIT = 'SETVAR <varname>=<varvalue>'
```

varname
The name of the installation variable to be set.

varvalue
The value that the installation variable is to be set to. If the value contains spaces, it must be enclosed in double quotes.

Example

The following example sets an installation variable named 'test' to 'This is a test'.

```
FILE
  EXIT = 'SETVAR test="This is a test"'
```

13.24 UPDATECONFIG

Use the UPDATECONFIG installation exit to update an environment variable value in the CONFIG.SYS file.

The UPDATECONFIG installation exit updates the CONFIG.SYS file only if the user has checked the **Update CONFIG.SYS** check box in the Install or Update window.

The syntax of this exit is:

```
FILE
  EXIT = 'UPDATECONFIG SET variable=newpath'
```

variable
The variable you are updating. If the variable already exists in the CONFIG.SYS file, then the value of newpath is added to the end of the existing variable. A semicolon is added between the existing variable and the newpath value.

If the variable does not exist in the CONFIG.SYS file, then a new variable is added at the end of the file.

newpath
The path you are adding to the variable in the CONFIG.SYS file.

The UPDATECONFIG installation exit attempts to match what is on the left side of the equals sign to the variables found in the CONFIG.SYS file. Therefore, if you are updating LIBPATH, remove SET from the exit. For example:

```
FILE
  EXIT = 'UPDATECONFIG LIBPATH=newpath'
```

Example

Installation variable substitution is useful in this exit. The following example shows how you can update the PATH and LIBPATH statements to contain the path your product was installed into.

```
*********************************************************
*            Update PATH                                *
*********************************************************
FILE
  EXITWHEN = 'INSTALL || UPDATE',
  EXITIGNOREERR = NO,
  EXIT = 'UPDATECONFIG SET PATH=%EPFIFILEDIR%'
*********************************************************
*            Update LIBPATH                             *
*********************************************************
FILE
  EXITWHEN = 'INSTALL || UPDATE',
  EXITIGNOREERR = NO,
  EXIT = 'UPDATECONFIG LIBPATH=%EPFIFILEDIR%'
```

Chapter 14. Servicing Your Shipped Product

,,,,,,,

Autolink to Sibling Window

RES='12100' VPX='25%' VPY='bottom' VPCX='75%' VPCY='100%

14.1 Specifying the PACKAGE Entry in the Catalog File

The installation and maintenance utility helps you service a product that is already installed. You can ship a corrective service that contains either the changed files only or a refresh of your entire product.

The way you specify the **PACKAGE entry [Link to heading 5.14 on page 22]** in the catalog file for corrective service depends upon whether the service is being shipped on a host tape or on diskettes.

Corrective services shipped on host tape

When corrective service is shipped on a host tape, the files on the tape are copied over the original file using SMP/E or some other tool that installs and manages products and services on a MVS system at your customer's shop. A VM EXEC is used for this copy on a VM system. This leaves a complete set of up-to-date files on the host at all times. Having a complete set of files available on the host gives a user who has already installed the product the ability to update to the new level. A user who has never installed the product can install the new level without having to install the old level.

Corrective services shipped on diskette

When corrective service is shipped on a diskette, a complete set of files might not be available. In this case, the corrective service diskettes (CSDs) should be used only by a user who already has the product installed. To prevent a user who has not yet installed a product from installing the CSDs, the UPDATE keyword in the PACKAGE entry in the product catalog file should be set to YES.

If a refresh of the entire product is shipped, the UPDATE keyword must be set to **NO**.

Regardless of how service is shipped, *it is very important that the package file contains a FILE entry for every file of the product, not just the files that are shipped*. If FILE entries are omitted, the delete and restore processes will fail.

The WHEN keyword should be used to specify when the installation utility transfers each file.

14.2 Things You Must Do

Prepare for shipping an corrective service of your product to your customers by doing the following:

1. Apply the correction to the product and rebuild it on the workstation.

   When the workstation build process uses the NMAKE utility, the output of the build contains only files that were affected by the correction. These affected files contain a new date and time.

2. Update the package files that contain FILE entries for the files that changed to reflect the new dates and times of the files. **Warning:** This is an important step because the dates and times are used by the installation and maintenance utility to determine which files to transfer during an update (if WHEN=OUTOFDATE in the FILE entries).

3. Update the SERVICELEVEL entry in the package file or files with the service level number for this correction.

   Changing the SERVICELEVEL entry notifies the installation and maintenance utility of the level of the code and gives the utility a way of tracking that service level.

4. If you are shipping on a host tape:

   a. Name the files the same as in the original shipment

   b. Put them into the same data sets for host installation

   c. Upload the files that have changed (or all files if you are shipping a refresh), including the package files, to the target host system

   d. Put them onto host tapes

5. If you are shipping on diskettes:

   a. Copy the files to the diskettes

   b. Update the VOLUME keywords of the FILE entry in the package file to reflect the volume label of the diskette the file was put on

   Note: Do not ship a new version of the catalog files unless this is corrective service being shipped on diskette.

6. Ship the tape or diskette with the corrective service to your customers.

Note: It is important that your documentation tells the end user to restart the operating system when told to. The restarting is necessary when changes have been made to the CONFIG.SYS file or when you have used the **REPLACEINUSE keyword of the FILE entry**[**Link to heading Chapter 11 on page 82**] in your package file.

14.3 Things Your Customer Must Do

Note: Users can Restore previous versions of your product only if they requested a backup version by selecting the **Save a backup version?** checkbox during the product update.

If the corrective service is on diskettes

The administrator is not involved. The user applies the correction to the workstation directly by using the following steps.

1. Insert the diskette containing the correction into the drive and perform an Open drive catalog procedure.

If the correction being applied is not a total refresh, the installation and maintenance utility ensures that the product is already installed before allowing the user to continue.

2. Select **Update** from the Action pull-down menu for the product being updated.

   The installation and maintenance utility transfers the changed files depending on the WHEN keyword value and records that a new service level has been applied on the workstation.

3. If the users find that the update contains errors or is incompatible with other programs, they can return their workstation copies of the product to the previous service level by selecting **Restore** from the Action pull-down menu.

If the corrective service is for MVS

1. The system administrator applies the correction to the original product on the host using SMP/E or another MVS installation manager.

   This updates the product data sets on the host with the new files. Files that were not shipped with the correction remain unchanged in the data sets.

2. The system administrator transfers the correction to the workstation using the installation and maintenance utility and tests the product.

3. The system administrator notifies the users that an update for the product is available.

4. The user starts the installation and maintenance utility and selects **Update** from the Action pull-down menu for the product that was updated. The installation and maintenance utility records that a new service level has been applied to the workstation.

   If the user find that the update contains errors or is incompatible with other programs, they can return their workstation copies of the product to the previous service level by selecting **Restore** from the Action pull-down menu.

If the corrective service is for VM

1. The system administrator applies the correction at the host using one of two options.

   Option 1 (preferred): Copy the files in the corrective service to a new minidisk. This is safer because the administrator will still have an unchanged copy of the original code if the correction is wrong.

   Option 2: Copy the files shipped with the correction onto the same minidisk as the original files. Doing this might cause a problem if the correction is wrong and the administrator wants to return the product code to its original state.

2. The system administrator transfers the correction to the workstation using the installation and maintenance utility and tests the product.

3. The system administrator notifies the users that an update for the product is available.

   If the administrator used option 2 to apply the correction, the end users need only link to the minidisk containing the code. If the administrator used option 1, the users must link to two minidisks, one containing the correction, and one containing the original code. The minidisk containing the correction must appear before the minidisk containing the original code in the VM search order.

   To make things easier for the user, after verifying that the correction is good, the administrator might want to copy the correction to the minidisk that contains the original code.

   After linking to the appropriate minidisks, the users can call the installation and maintenance utility and select **Update** from the Action pull-down menu for the product being updated. The installation and maintenance utility records that a new service level has been applied to the workstation.

If users find that the update contains errors or is incompatible with other programs, they can return their workstation copy of the product to the previous service level by selecting **Restore** from the Action pull-down menu.

Chapter 15. Renaming Software Installer Files

''''

Autolink to Sibling Window

RES='&hres.' VPX='25%' VPY='bottom' VPCX='75%'

15.1 Avoiding Name Conflicts in Redistributed Files

Before you begin creating the files to install your product, rename all installation and maintenance utility files that you redistribute. This renaming ensures that other applications using installation and maintenance utility files are not in conflict with yours.

You should use the command file ISREN.CMD to rename installation and maintenance utility files you are redistributing and place them in a directory you specify. Do not use the same directory you are renaming from to rename to. If the directory you specify does not exist, it will be created. You should use this as your working directory for enabling your product to use installation and maintenance utility.

The **"Step-by-Step Instructions for Using Software Installer" [Link to heading 3.7 on page 10]** tells you how to use your customized copy of Software Installer.

Follow these steps to rename the installation and maintenance utility files you want to redistribute.

1. Choose a three-letter prefix to identify your product's redistributed installation and maintenance utility files.
2. Choose the output directory where you want the renamed files. This will be your working directory for creating your installation files and customizing installation for your product.
3. Switch to the \IBB\BIN directory that contains the installation and maintenance utility files.
4. Start ISREN.CMD.
5. If you do not specify a three-character prefix or an existing directory as parameters of the rename command, you will be prompted for them.
6. You are notified on the screen that the installation and maintenance utility files will be renamed and placed in the directory you specified.
7. If the prefix and directory confirmation are correct, press the Enter key.
8. If you want to stop the renaming process, enter any character.
9. Progress is displayed on your screen as each file is renamed.
10. Check the specified directory for the renamed files.

You are now ready to build the files for your product's installation.

15.2 Renaming .CMD and Parameters

The command for renaming installation and maintenance utility files is:

ISREN [prefix [output-directory]]

Where:

prefix
Specifies the product prefix that you want the renamed component to contain. If this parameter is omitted or is not exactly three characters in length, you are prompted to enter a prefix. If you enter the prefix in lower case, the prefix will appear in lower case in the renamed file names.

output-directory
Specifies the directory where the renamed copies of the files are to be placed. If this parameter is not specified, you are prompted to enter the directory. If this directory does not exist, it is created for you.

Chapter 16. Automating Installation of Your Product

,,,,,,,

Autolink to Sibling Window

RES='22300' VPX='25%' VPY='bottom' VPCX='75%' VPCY='100%

16.1 Methods of Automating Installation of Your Product

There are two ways to pass information directly to the installation and maintenance utility:

- **EPFINSTS with parameters from the command line** [**Link to heading 16.2 on page 107**]

  If you have **included FILE entries for the Software Installer files in your product's package file**[**Link to heading 6.3 on page 28**], the user of your product can start the installation process from an OS/2 command line by using the renamed EPFINSTS command with parameters. Starting the installation by using the renamed EPFINSTS.EXE takes the user to the catalog list and bypasses the initial bit maps.

  The end user can also use **command line parameters with INSTALL.EXE and EPFIDLDS.EXE**[**Link to heading 16.3 on page 112**].

- **Response files** [**Link to heading 16.4 on page 113**]

  You can provide response files that contain the needed information to install your product and to make necessary changes to the target system's configuration files during an unattended installation. This information would otherwise be provided by the user in an attended installation scenario.

16.2 EPFINSTS with Parameters from the Command Line

You can use the renamed EPFINSTS command followed by one or more parameters at the OS/2 command prompt to pass information to the installation and maintenance utility.

If no parameters are specified with the renamed EPFINSTS command, the installation will be interactive. To install non-interactively, the user should use the /X parameter following the renamed EPFINSTS command. This causes the installation and maintenance utility to use the information you have supplied in the response files.

If the value of any of the parameters contains spaces, use double quotes to enclose the value.

Values can be upper or lower case. Parameters can be in any order.

You can use the = or the : format in the following syntax:

```
EPFINSTS /A:<action>
         /C:<catalog file name>
         /F:<response file>
         /G:<include path>
         /L:<xpos,ypos>
         /L1:<error log>
         /L2:<history log>
         /L3:<log file>
         /L4:<log file>
         /L5:<log file>
         /O:<originating system>
         /P:<product name>
         /R:<response file>
         /S:<source location>
         /T:<install target directory>
         /TU:<update CONFIG.SYS directory>
         /X
```

The following parameters are required for a **CID**[**Link to heading Chapter 20 on page 123**] enabled installation.

/A Action

/C Product catalog name

/P Product name

/O Originating system if action is install or update

Related topics

Definition of Parameters for EPFINSTS [**Link to heading 16.2.1 on page 108**]
Examples of EPFINSTS with Parameters [**Link to heading 16.2.2 on page 111**]

16.2.1 Definition of Parameters for EPFINSTS

The EPFINSTS command line parameters are:

/A:<action>
Specifies the action to be performed.

If you use this parameter, the main window of the installation and maintenance utility is not displayed.

If you do not use this parameter, the installation and maintenance utility starts normally with all windows displayed.

If you specify this parameter, you must also specify the /P parameter. Valid values for this parameter are:

'D' For delete
'I' For install
'R' For restore
'U' For update

The installation and maintenance utility sets the EPFIACTION installation variable to the action.

/C:<catalog file name>
Specifies the name and location of the catalog file that you want to be opened automatically.

If the catalog file is located on a host session, the host session ID must precede the catalog file name. If the catalog file is located on a drive, a drive and directory must precede the catalog file name.

The installation and maintenance utility sets the EPFICATALOG installation variable to the catalog file name.

The following are examples of this parameter for each installation source.

- For Drive: /C:A:DISK1.ICF
- For MVS: /C:B:EPF.V1R2M0.SEPFBENU(EPFICAT)
- For VM: /C:"B:EPFICAT ENUEPF12 *"
- For VSE: /C:B:LIB.SUBLIB.MEMBERNAME.MEMBERTYPE

If only the host session ID is specified, for example, B: then the Open catalog window is displayed with **B:** as the default session or drive.

/F:<response file>
Specifies the specific response file on the workstation that is read for **additional information**[**Link to heading 16.4 on page 113**].

The drive and path are optional. The following search order is used to find the response files.

1. The fully qualified file specification
2. The current directory
3. The file name together with the /G: invocation parameter
4. Each directory in the PATH environment variable
5. Each directory in the DPATH environment variable

The function of /F: is the same as /R:. If both are specified, /R: takes precedence over /F:.

The installation and maintenance utility sets the EPFIRESPFILE installation variable to the response file.

/G:<include path>
Specifies the drive and directory of the general response files that are included by the specific response file.

The installation and maintenance utility sets the EPFIGENRESPDIR installation variable to the included path.

Example:

/G:L:\XYZ

/L:<xpos,ypos>
Specifies the initial horizontal and vertical coordinates of the installation and maintenance utility window on the Workplace Shell* relative to the lower-left corner of the screen.

If this parameter is specified, all previously saved default positions are overridden.

The installation and maintenance utility sets the EPFIWNDCOORDS installation variable to the coordinates of the installation and maintenance window.

/L1:<error log>
Specifies the drive, path, and file name of the error log file.

The installation and maintenance utility sets the EPFIERRORLOG installation variable to the name of the error log.

If drive and path are not specified, the drive and path where EPFINSTS.EXE is running is used. If this parameter is not specified, errors are logged to EPFINSTS.OUT. If the /X parameter is not used, no error log file is created because the installation is interactive and errors are displayed.

Example of using the /L parameter to create ERROR.LOG file in the C:\ABC\ directory:

/L1:C:\ABC\ERROR.LOG

/L2:<history log>
Specifies the drive, path, and file name of the history log file.

The installation and maintenance utility sets the EPFIHISTORYLOG installation variable to the name of the history log.

If drive and path are not specified, the drive and path where EPFINSTS.EXE is running is used.

If this parameter is not specified, no history log is maintained.

Example:

/L2:C:\ABC\HISTORY.LOG

/L3: /L4: /L5:<log files>
Each of these parameters can contain a drive, path, and file name of a log file. They are preserved in installation variables EPFIL3LOG, EPFIL4LOG, and EPFIL5LOG for your use.

/O:<originating system>
Specifies the originating system of the installation. Valid values are:

- DRIVE
- MVS
- VM
- VSE

The installation and maintenance utility sets the EPFISOURCE installation variable to one of these values.

/P:<product name>
Provides the name of the product to perform the action on.

If you do not use this parameter, the installation and maintenance utility comes up normally with all windows displayed.

The <product name> should match the value of the NAME keyword of the PACKAGE entry in the catalog file. If your product name string includes any spaces, use double quotes around the string. For example:

/p:"product name with spaces"

The installation and maintenance utility sets the EPFIPRODUCT installation variable to the name of the product.

/R:<response file>
Specifies the drive, path, and file name of the specific response file.

The function of the /R: parameter is the same as the /F: parameter. If both are specified, /R: takes precedence over /F:. If neither /F: nor /R: is used, no response files are used.

The following search order is used to find the response files.

1. The fully qualified file specification
2. The current directory
3. The file name together with the /G: invocation parameter
4. Each directory in the PATH environment variable
5. Each directory in the DPATH environment variable

The installation and maintenance utility sets the EPFIRESPFILE installation variable to the response file.

Example:

/R:L:\XYZ\RESPONSE.DAT

/S:<source location>
Specifies the drive and path that contain the source files to be installed.

The installation and maintenance utility sets the EPFISOURCEDIR installation variable to the source location.

Specifying this parameter overrides the value of the HOSTLOC keyword in the product's catalog file PACKAGE entry.

If this parameter is not specified, the value of the HOSTLOC keyword is not overridden.

Example:

```
/S:L:\XYZ
```

/T:<install target directory>
Specifies the drive and path that the product files are installed into.

This parameter only affects the value of the FILE directory (directory corresponding to the FILELABEL keyword). If this parameter is specified, it will override the FILE paths that are specified in the response files and the package files.

The installation and maintenance utility sets the EPFITARGETDIR installation variable to the install target directory.

Example:

```
/T:C:\IBB
```

/TU:<update target CONFIG.SYS directory>
Specifies the drive and path of the target CONFIG.SYS to be updated.

The installation and maintenance utility sets the EPFITARGETCNF installation variable to the update CONFIG.SYS target directory.

If this parameter is not specified, the CONFIG.SYS files are updated as specified in the product's package file.

Example:

```
/TU:C:\
```

/X
Specifies that the action is noninteractive.

If you do not specify all of the information needed for the action to complete, an error occurs. When you specify this option, no progress indication is shown and all error messages are logged in the EPFINSTS.OUT file. You can specify the location of this error log file by using the /L1 parameter.

If you do not specify this option, the user is prompted for any information that the installation and maintenance utility needs to complete the action. In this interactive mode of action, progress indication is shown and error messages are displayed to the user in secondary windows.

The installation and maintenance utility sets the EPFINONINTERACT installation variable to YES if the /X parameter is specified. It is set to NO if the /X parameter is not specified.

16.2.2 Examples of EPFINSTS with Parameters

The following two examples show a drive source and a host source installation from the command line using the EPFINSTS command and parameters.

Drive system source

The /X makes this a noninteractive installation.

```
EPFINSTS /A:I
         /C:B:\XYZ\PRODUCT.ICF
         /G:L:\XYZ
         /L1:M:\ERROR.LOG
         /L2:M:\HISTORY.LOG
         /O:DRIVE
         /P:"XYZ PRODUCT"
         /R:L:\XYZ\RESPONSE.DAT
         /S:L:\XYZ
         /T:C:\IBB
         /TU:C:\
         /X
```

Host system source

```
EPFINSTS /O:MVS /C:B:EPF.V1R2M0.SEPFBENU(EPFICAT) /A:I /P:"MY PRODUCT" /R:RESPFILE.DAT
```

This command opens the EPFICAT catalog file that is located on an MVS system in the B emulator session. Because this is an installation (/A=I), the package file is transferred from the host. The installation and maintenance utility looks for any additional parameters in the RESPFILE.DAT **response file**[**Link to heading 16.4 on page 113**].

16.3 Using Parameters with INSTALL.EXE and EPFIDLDS.EXE

The user installing your CID enabled product can use the following **parameters** [**Link to heading 16.2.1 on page 108**] on the command line with the INSTALL.EXE or renamed EPFIDLDS.EXE.

/A: <action>

/G: <include path>

/L1: <error log>

/L2: <history log>

/L3: <log file>

/L4: <log file>

/L5: <log file>

/O: <orginating system>

/R: <response file>

/S: <source location>

/T: <target directory>

/TU: <update CONFIG.SYS directory>

/X: interactive flag

Note: The drive/directory/filename path is required for all log parameters specified with the INSTALL or EPFIDLDS commands. If the full path is not specified, the logs will be created in a temporary directory that is deleted when processing is completed.

You should advise your users to use these parameters when they are installing your product differently than you have specified in your IIRC.RC resource file.

The values specified from the command line override those you specified when you modified the **IIRC.RC**[**Link to heading 8.1 on page 71**].

Parameter	Overrides
/A	STARTPARM_ACTION
/O	INST_SOURCES
/R	STARTPARM_RESPFILE
/T	INST_DESTINATION
/S	The use of the /S parameter is affected by the /O parameter and the INST_SOURCES value specified in the IIRC.RC file. If the end user doing a noninteractive (/X) installation does not use the /O parameter and you have more than one source specified in your IIRC.RC file, Software Installer displays an error message telling the user that the /O parameter is required. If the /O the user specifies a host source system, the /S parameter can override the MVS_SOURCE_DSN and the VSE_SOURCE_NAME. For MVS and VSE /S:B:<user_mvs_data_set_name> would override the MVS and VSE source parameters specified in the IIRC.RC for the user's B: session. For VM, /S:B: tells Software Installer the source is on the B: session when /O is specified as VM. In other words, when /O specifies a host system, /S specifies the host session.

16.4 Response File Usage

You can use **response files**[**Link to heading 16.6 on page 115**] to **pass information**[**Link to heading 16.5 on page 114**] to the installation and maintenance utility.

Software Installer supports installations that have one specific response file and optionally have general response files that are included by the specific response file.

The syntax for the lines in a response file is:

```
keyword = value
```

Keyword-value pairs used in a response file can be in any order. However, there can be only one pair per line.

If you use any keyword names other than the supported keywords, they will be treated as installation variables. For example, MONITOR = VGA would create a installation variable with the name of MONITOR and a value of VGA.

An example of a response file that can be used to customize an installation is:

```
FILE      = D:\PRODUCT\EXE\LIB
WORK      = D:\PRODUCT\WORK
COMP      = COMPONENT1_NAME
COMP      = COMPONENT2_NAME
COMP      = COMPONENT3_NAME
CFGUPDATE = AUTO
OVERWRITE = YES
```

Note: Do not use quotes around the component name (even when the name is more than one word with blanks between words).

16.5 Keywords for Response Files

Software Installer supports the following keywords:

AUXn
Specifies the new default path for the auxiliary directory, where **n** is any number between 1 and 18. (Thus there are 18 possible auxiliary directories: AUX1, AUX2,...AUX18.) This AUXn value is used in place of the PATH entry value of the AUXn keyword. This keyword is used only for installation processing.

CFGUPDATE
Specifies whether the CONFIG.SYS file is automatically updated. Valid values for this keyword are:

AUTO Automatically updates CONFIG.SYS

MANUAL Does not update CONFIG.SYS

COMP
Specifies the unique name of a component of the product for which passed information applies. You can use a maximum of 50 COMP keywords in a response file.

Note: Do not use quotes around the component name (even when the name is more than one word with blanks between words).

COPY
Specifies the source and target files for a copy process. The format of this keyword is:

```
COPY = source_filespec target_filespec
```

If the target_filespec already exists, it is overwritten. If either file specification is not valid, the copy is not performed.

DELETEBACKUP
Specifies whether to delete only the backup versions of the product or to delete the entire product. Valid values for this keyword are YES and NO.

FILE
Provides the new default path for the file directory. This FILE value is used in place of the PATH entry value of the FILE keyword. This keyword is used only for an installation action.

INCLUDE
Specifies which general response files to include with a specific response file. The format of this keyword is:

```
INCLUDE = filespec
```

Where filespec is the general response file to be included. If the file specification contains any global characters (* or ?), the first file found which matches the specification is included. If the specification is not valid, no general response file is included.

Note: You should not have more than five levels of included response files.

The following represents the maximum permitted five levels of response file inclusion:

```
LEVEL1.RSP includes LEVEL2.RSP
  LEVEL2.RSP includes LEVEL3.RSP
    LEVEL3.RSP includes LEVEL4.RSP
      LEVEL4.RSP includes LEVEL5.RSP
```

The following search order is used to find the general response files specified.

1. The fully qualified file specification, if specified with filespec
2. The current directory
3. The file name together with the /G: invocation parameter

4. Each directory in the PATH environment variable
5. Each directory in the DPATH environment variable

OVERWRITE
Specifies whether to automatically overwrite files during installation. Valid values for this keyword are YES and NO.

SAVEBACKUP
Specifies whether to save a backup version of the product when it is updated. Valid values for this keyword are YES and NO.

USEREXIT
Specifies the name of an exit that you want started. The format of this keyword is:

USEREXIT = filespec

Where filespec is the name of a user exit. If the file specification contains any global characters (* or ?), the first executable found which matches the specification is started. If the specification is not valid, no user exit is started.

The following search order is used to find the specified user exit.

1. The fully qualified file specification, if specified with filespec
2. The current directory
3. Each directory in the PATH environment variable
4. Each directory in the DPATH environment variable

WORK
Provides the new default path for the data directory. This WORK value is used in place of the PATH entry value of the WORK keyword. This keyword is used only for an installation action.

16.6 Response File Details

A response file is a flat ASCII file that consists of a series of lines separated by newline sequences (0xa, 0x0d, or a combination of these two sequences).

Each line in a response file has a maximum line length of 255 bytes.

There are two kinds of lines in a response file:

Comment lines
Comment lines contain only white space characters, or have either an asterisk (*) or a semicolon (;) as the first non-white space character on the line.

Response lines
Response lines are used by Software Installer to determine the options and configurations to install on the target system.

Response lines have the following syntax:

keyword = value

Keywords cannot contain imbedded spaces.

Keywords are not case-sensitive.

Keywords can be grouped together in value lists using the following syntax:

```
keyword = (
            keyword1 = value
            keyword2 = value
            .
            .
            keywordn = value
          )
```

16.7 Return Codes

Software Installer returns the following return codes to a **software distribution manager (SDM)**[**Link to heading Chapter 27 on page 130**].

Messages are logged only when the /X parameter is used with the EPFINSTS command.

All return codes are two-byte hexadecimal values.

00 00 Successful Software Installer termination. No messages were logged.

00 04 Successful Software Installer termination. Warning messages were logged. Restarting the workstation operating system is not required.

00 08 Successful Software Installer termination. Error messages were logged. Restarting the workstation operating system is not required.

00 12 Successful Software Installer termination. Severe error messages were logged. Restarting the workstation operating system is not required.

16 00 Software Installer received an incorrect command line parameter. Messages were logged. Correct the parameter and try again.

16 04 Software Installer encountered an unexpected condition. Messages were logged.

FE 00 Successful Software Installer termination. Restarting the workstation operating system is required. Do not invoke Software Installer again.

FE 04 Successful Software Installer termination. Warning messages were logged. Restarting the workstation operating system is required. Do not invoke Software Installer again.

FE 12 Successful Software Installer termination. Severe error messages were logged. Restarting the workstation operating system is required. Do not invoke Software Installer again.

FF yy Successful Software Installer termination. *yy* can vary from 00 to FF. Restarting the workstation operating system is required. Do not invoke Software Installer again.

Your application specific exits that run during processing of you package file should return the following return codes. You can use the C language EXIT() function to return these codes. Software Installer intercepts the return codes and displays the appropriate message to the user if you have set the value of the **EXITIGNOREERR keyword**[**Link to heading 6.28 on page 49**] to NO. No user message is displayed if EXITIGNOREERR is YES.

All return codes are two-byte hexadecimal values. *xx* can vary from 00 to FF.

00 00 Successful exit termination.

FE xx Return this code when your product exit processing requires restarting the workstation operating system without starting the installation processing. Software Installer stops processing your product package file immediately when this code is received. User message EPFII177 is displayed: 'The action you requested has completed. You must restart the operating system.'

FF xx Return this code when your product exit processing requires restarting the workstation operating system and restarting the installation processing. Software Installer stops processing your product package file immediately when this code is received. User message EPFII178 is displayed: ′The action you requested has stopped. You must restart the operating system and retry the action.′

If a non-CID return code is returned to Software Installer, user message EPFIE187 is displayed: ′A product-specific installation error occurred while executing the <variable> exit routine. The return code is <variable>.′

Chapter 17. Installation Variables

For more information on passing parameters to installation exits, see **Passing Data Between Software Installer and Exits**[**Link to heading 13.5 on page 86**].

If your installation specifies HPFS files or directories, enclose the file or directory variables with double quotes. The double quotes are necessary to preserve any spaces in the HPFS names.

Examples in this document are for FAT file systems and do not have the double quotes enclosing variable names.

The following installation variables are available during the installation.

EPFIACTION
Action being performed by the installation and maintenance utility (DELETE, INSTALL, RESTORE, or UPDATE).

EPFIAUXnDIR
AUXn directory that the user specifies in the Install - directories window. The n is a number between 1 and 18. See the description of the PWSPATH keyword for the package file **FILE entry type** [**Link to heading 6.28 on page 49**] for an explanation of the directory entry fields in the Install - directories window.

EPFIBOOTDRIVE
Drive that the operating system was started from. For example, if OS/2 was started from the C: drive, this variable is set to C.

EPFICATALOG
Value of the /C parameter of the **EPFINSTS command**[**Link to heading 16.2 on page 107**].

EPFICOMPNAME
Name of the component currently being processed. This string comes from the NAME keyword of the COMPONENT entry in the package file.

EPFICOMPSIZE
Size of the component currently being processed.

EPFICURHOST
Source name of the file currently being processed. This file name does not include the drive or session. It does include the directory, data set name, or minidisk. For example, if the SOURCE keyword of a FILE entry is:

SOURCE=' DRIVE: filename.ext; VM: filename ext *'

then the value of EPFICURHOST is \filename.ext when this package is installed from a drive.

EPFICURNAME
Name of the file currently being processed.

EPFICURPWS
Fully qualified name of the workstation file currently being transferred.

EPFICURPWSDIR
Fully qualified directory that the file currently being processed will be transferred to.

EPFICURSIZE
Size of the file currently being processed.

EPFICURUPD
The fully qualified file name that the file currently being processed will be unpacked into.

EPFICURUPDIR
The fully qualified directory that the file currently being processed will be unpacked into.

EPFICURUPS
The fully qualified name of the source file currently being unpacked.

EPFIERRORLOG
The drive, path, and file name of the error log file.

EPFIFILEDIR
Directory where the product is being installed. This is the FILE directory that the user specifies in the Install - directories window. See the description of the PWSPATH keyword for the package file **FILE entry type** [**Link to heading 6.28 on page 49**] for an explanation of the directory entry fields in the Install - directories window.

EPFIGENRESPDIR
The drive and path of the general response files that are included by the specific response file.

EPFIHISTORYLOG
The drive, path, and file name of the history log file.

EPFIL3LOG
The drive, path, and file name of a log file.

EPFIL4LOG
The drive, path, and file name of a log file.

EPFIL5LOG
The drive, path, and file name of a log file.

EPFIOS
Current operating system. In Software Installer, the value of this variable is **OS2**.

EPFIOVERWRITE
Indicator of whether the workstation files will be overwritten. Valid values are:

YES The user selected the **Overwrite files** option in the Install or Update window.

NO The user did not select the overwrite option.

EPFINONINTERACT
Evaluates to YES if the /X parameter of the **EPFINSTS command**[**Link to heading 16.2 on page 107**] is specified. Evaluates to NO if /X is not specified with the EPFINSTS command.

EPFIPKGDESCR
Value of the PACKAGE entry PKGDESCRFILE keyword in the catalog file.

EPFIPKGFEATURE
Value of the PACKAGE entry FEATURE keyword in the catalog file.

EPFIPKGFILE
Value of the PACKAGE entry PACKAGEFILE keyword in the catalog file.

EPFIPKGNAME
Value of the PACKAGE entry NAME keyword in the catalog file.

EPFIPKGNUMBER
Value of the PACKAGE entry NUMBER keyword in the catalog file.

EPFIPKGSVCLVL
Value of the SERVICELEVEL entry LEVEL keyword in the package file.

EPFIPKGUPDATE
Value of the UPDATE keyword for the PACKAGE entry. It specifies whether all files are shipped or just the files changed since the previous shipment. Valid values are YES and NO.

EPFIPKGVRM
Value of the PACKAGE entry VRM keyword in the catalog file. The value specifies the version, release, and modification number of a product.

EPFIPRODUCT
Value of the /P parameter of the **EPFINSTS command**[**Link to heading 16.2 on page 107**].

EPFIRESPFILE
Value of the /F parameter of the **EPFINSTS command**[**Link to heading 16.2 on page 107**].

EPFISESSION
Host session or drive used for file transfer (such as A, B, or C).

EPFISOURCE
Originating system of source files. Valid values are: MVS, VM, VSE, or DRIVE. This value can be overridden with the /O parameter.

EPFISOURCEDIR
Drive and directory that contains the source files. This value can be overridden with the /S parameter.

EPFITARGETCNF
Drive and directory containing the target CONFIG.SYS file.

EPFIUPDATECNF
Indicator to update the configuration file. Valid values are:

YES The user selected the **Update CONFIG.SYS** option in the Install or Update window.

NO The user did not select the update option.

EPFIWNDCOORDS
Value of the /L parameter of the **EPFINSTS command**[**Link to heading 16.2 on page 107**].

EPFIWORKDIR
Directory for installation processing. This is the work directory that the user specifies in the Install - directories window. See the description of the PWSPATH keyword for the package file **FILE entry type** [**Link to heading 6.28 on page 49**] for an explanation of the directory entry fields in the Install - directories window.

Chapter 18. EPFINSTDIR Environment Variable

You can use the environment variable EPFINSTDIR to specify the directory where files used by the installation and maintenance utility are copied.

You can set this variable by having the end user set it with the OS/2 SET command before starting installation or have the end user put a SET command in the CONFIG.SYS file and restart the system.

If you do not set this environment variable, the files are stored in the \OS2\SYSTEM directory on the drive where the operating system is started.

You may want set this variable if space is a consideration in your \OS2\SYSTEM directory or if you want to install from a server onto multiple client workstations. The installation and maintenance utility stores the EPFINS.INI in the directory specified by EPFINSTDIR. By changing EPFINSTDIR, you change where EPFIS.INI is stored. This means that if the installation utility can't find EPFIS.INI it allows the product to be installed multiple times.

You cannot run the installation from a server and install products on multiple clients. However, the following steps enables multiple clients to install your product from a server using entries in your package file to create Workplace Shell objects.

Use a .CMD file to :

1. Link the server to the client
2. Have the client set the EPFINSTDIR using:

   SET EPFINSTDIR=<client_drive_name>:\

3. Have the client start the installation that is on the server
4. When installation is complete, break the link to the client and go to the next one

For more information on passing parameters to installation exits, see **Passing Data Between Software Installer and Exits**[**Link to heading 13.5 on page 86**].

Chapter 19. Glossary

This is a glossary of Software Installer terms.

CID [**Link to heading Chapter 20 on page 123**]
drive [**Link to heading Chapter 21 on page 124**]
entry [**Link to heading Chapter 22 on page 125**]
global character [**Link to heading Chapter 23 on page 126**]
hook [**Link to heading Chapter 24 on page 127**]
installation exit [**Link to heading Chapter 25 on page 128**]
product prefix [**Link to heading Chapter 26 on page 129**]
SDM [**Link to heading Chapter 27 on page 130**]

Chapter 20. CID (definition)

Configuration, Installation, and Distribution (CID) is IBM′s term for capabilities to automate installation. CID enabled products will be capable of unattended, remote installation.

Chapter 21. drive (definition)

A drive can be a diskette drive, a LAN drive, or an AS/400 PC Support folder.

Chapter 22. entry (definition)

A record in a catalog file or a package file. An entry consists of an entry type and one or more keywords and their corresponding values. These entry types and their keywords provide input information to the installation and maintenance utility about the product that is to be installed.

Chapter 23. global character (definition)

Characters that allow you to search for nonspecific character strings. Global characters (sometimes called ″wildcards″) free you from specifying the exact characters in a field.

You can use an asterisk or a question mark as a global character. An asterisk denotes a sequence of characters; a question mark denotes a single character.

For example, you can type AB* to search for all entries beginning with the letters AB.

Chapter 24. hook (definition)

A mechanism by which procedures are called when certain events occur in the system.

Chapter 25. installation exit (definition)

An EXE, DLL, or CMD file that runs outside of installation processing (sometimes called ″user exit″).

Chapter 26. product prefix (definition)

The unique three-character identifier you choose that begins all part names of a Software Installer-enabled product. The prefix is used to rename installation and maintenance utility files that are shipped as part of other products. Renaming prevents file name conflicts with other products that use the installation and maintenance utility files.

Chapter 27. SDM (definition)

Server systems in the **CID** [**Link to heading Chapter 20 on page 123**] environment are referred to as software distribution managers (SDM). In a CID environment the SDM calls Software Installer.

Table 4. Footnote Panels for Current Heading

Footnote ID	Footnote
drivfn	A drive can be a diskette drive, a LAN drive, or an AS/400 PC Support folder.

Chapter 28. Index

E

F

G

H

I

K

L

M

O

P

R

S

T

U

W