VisualAge Pacbase

PAW
Developer's Guide

DSPWO000161A

Note

Before using this document, read the general information under "Notices" on the next page.

According to your license agreement, you may consult or download the complete up-to-date collection of the VisualAge Pacbase documentation from the VisualAge Pacbase Support Center at:
http://www.software.ibm.com/ad/vapacbase/support.htm

Consult the Catalog section in the Documentation home page to make sure you have the most recent edition of this document.

1st Edition (July 1997)

This edition applies to the following licensed program:

- VisualAge Pacbase Versions 2.0 and 2.5

Comments on publications (including document reference number) should be sent electronically through the Support Center Web site at:
http://www.software.ibm.com/ad/vapacbase/support.htm
or to the following postal address:
IBM Paris Laboratory
VisualAge Pacbase Support
30, rue du Château des Rentiers
75640 PARIS Cedex 13
FRANCE

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

NOTICES

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Subject to IBM's valid intellectual property or other legally protectable rights, any functionally equivalent product, program, or service may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

Intellectual Property and Licensing
International Business Machines Corporation
North Castle Drive, Armonk, New-York 10504-1785
USA

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of information which has been exchanged, should contact:

IBM Paris Laboratory
SMC Department
30, rue du Château des Rentiers
75640 PARIS Cedex 13
FRANCE

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

IBM may change this publication, the product described herein, or both.

TRADEMARKS

IBM is a trademark of International Business Machines Corporation, Inc.
AIX, AS/400, CICS, CICS/MVS, CICS/VSE, COBOL/2, DB2, IMS, MQSeries, OS/2, PACBASE, RACF, RS/6000, SQL/DS, TeamConnection, and VisualAge are trademarks of International Business Machines Corporation, Inc. in the United States and/or other countries.

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States and/or other countries.
Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States and/or other countries.
UNIX is a registered trademark in the United States and/or other countries licensed exclusively through X/Open Company Limited.

All other company, product, and service names may be trademarks of their respective owners.

Table of Contents

1. Presentation of the Developer's and User's Workstations 9
1.1. Developer's Workstation 9
1.2. User's Workstation 10

2. Installation 13
2.1. Installation Procedure 13
2.1.1. Common Characteristics of Dialogue Boxes 13
2.1.2. 'Installation Language' Box 13
2.1.3. 'PAW user language' box 13
2.1.4. 'ENVIRON.PRM Directory' Box 14
2.1.5. 'PAW Installation Parameters' Box 14
2.1.5.1. 'Workstation Type' 14
2.1.5.2. 'Learning Mode' 14
2.1.5.3. 'Root Directory' 15
2.1.5.4. 'Unit' 16
2.1.5.5. 'Disk Space' 16
2.1.6. 'PAW Component Directories' Box 16
2.1.7. 'Windows Directory' Box 17
2.1.8. 'Communication Protocol' Box 17
2.1.9. 'Protocol Variant' Box 17
2.1.10. 'Reinstalling PAW on a Previous Version' Box 17
2.1.11. End of Installation 17
2.1.11.1. Developer's Workstation 18
2.1.11.2. User's Workstation 18
2.2. Customization of the Workstation 18
2.2.1. ENVIRON.PRM file 18
2.2.2. PAWLIB.PRM file 21
2.2.3. PAWMAK.PRM file 22

3. Revamping an Application 25
3.1. Extraction from the Server 26
3.2. Transferring the Extraction File to a PC 26
3.3. Generating the Screen Source Files 26
3.4. Compilation 27
3.5. Tests on the Generation and Compilation Steps 28
3.6. Implementation 30

4. Error Management 31
4.1. Installation Errors 31
4.2. Generation Errors 31
4.3. Compilation Errors 33
4.4. Operation Errors 33

5. Advanced Functions 37
5.1. External Value Lists 37
5.1.1. Structure of External Value Source Files 38
5.1.2. Local Generation of the External Value Lists 39
5.1.3. Compiling and Checking the Resulting Files 39
5.1.4. To put the Characteristics of a Data Element into Contact and to Enrich them. 40
5.1.5. Generation of Revamping Files by PAWGEN 42
5.2. Customizing the On-Line Help 43
5.2.1. Modifying an Existing Text 43
5.2.2. Adding One or Several Divisions to a Help Text 44
5.2.3. Regenerating the On-Line Help 44
5.3. Automating the Tasks: .BAT 45
5.4. Keyboard Configuration 47
5.4.1. Generalities 47
5.4.2. Local Functions 49
5.4.3. Values of Keys (to be inserted in PAW_KBRD.PRM) 50
5.5. ClickPad 50

5.5.1. Characteristics 50
5.5.2. Changing the Standard Icons 51
5.5.2.1. Adding New Icons in the ICOPAW.RC File 51
5.5.2.2. Changing the Icon of a Standard Button 52
5.5.2.3. Associating an Icon with an Action or Screen Branching Button 52
5.5.2.4. Associating an Icon with a Script 52

6. Examples of PAW revamping 53
6.1. Porting a MICROFOCUS Application onto a Revamped Application 53
6.1.1. Architecture of a DOS MICROFOCUS Dialogue Application 53
6.1.2. Architecture of the Revamped Application under WINDOWS 3 54
6.1.3. Notes and Recommendations 54
6.1.4. Examples of Compilation Command Files 56
6.2. Revamping an IBM Product: DSMS 57
6.2.1. DSMS Revamping: Introduction 57
6.2.2. Installation 58
6.2.3. Building the External Value Lists 59
6.2.4. Keyboard Configuration 60
6.2.5. If problems arise 60

7. PAW DDE Server 61
7.1. Characteristics of a DDE Dialogue 61
7.2. Characteristics of DDE Connections 62
7.3. Syntax of the Items 63
7.4. Syntax of the Commands 64
7.5. VisualBasic Example of PAW used as a DDE Server 65

8. Script Language 67
8.1. Introduction 67
8.2. Types of Scripts 68
8.3. Implementation of Scripts 69
8.3.1. Parameters 69
8.3.1.1. Line Structure of the Scripts 69
8.3.1.2. List of Logon Scripts 69
8.3.1.3. List of Application and Screen Scripts 70
8.3.2. Scripts and DDE 70
8.4. Script Structure 70
8.5. Reserved Words 71
8.6. Declarations 71
8.6.1. Variable Types 71
8.6.2. Variable Names 72
8.6.3. Variable Values 72
8.6.3.1. Test Syntax of a Boolean 73
8.6.4. Labels 74
8.6.5. Comments 74
8.6.6. Blanks and Returns 74
8.7. The Body of the Program 75
8.8. Instructions 75
8.9. Assignments 75
8.10. Expressions and Operators 76
8.10.1. Priority of Operators 76
8.10.2. Processing of Associative Operators 77
8.11. The Unconditional Branching 77
8.12. Control Structures 77
8.13. Expressions 78
8.14. Functions 79
8.14.1. Function Call 79
8.14.2. Function Parameters 80
8.15. Errors 95
8.15.1. Source Code Errors 95
8.15.2. Syntax Errors 95
8.15.3. Errors during Execution 96

Presentation of PAW

PAW (**P**acbench **A**utomatic **W**indowing) is an application designed to provide PC-revamping for your mainframe applications and to help you use them with greater comfort.

It is based on a MS-WINDOWS-type graphical interface, offering typical facilities such as: mouse-support, pop-up menus, and on-line help).

The terms "user" and "developer" are used in this manual with the following meanings:

- **the user** is the person who uses the application. He/she works on a personal computer on which PAW has been installed, i.e. on which the developer has set up the revamping parameters. The module used by the user is called PAW.

- **the developer** is the person in charge of installing the program, realizing the application revamping, i.e. generating the revamping parameter files, and installing them on the users' Pcs. The modules used by the developer to generate the revamping parameter files are called PAWGEN and PAWLIS.

Note concerning this manual

This manual is designed for two types of customers: those who want to revamp applications that they have developed, and those who want to revamp an application originally designed for a dumb terminal, such as DSMS.

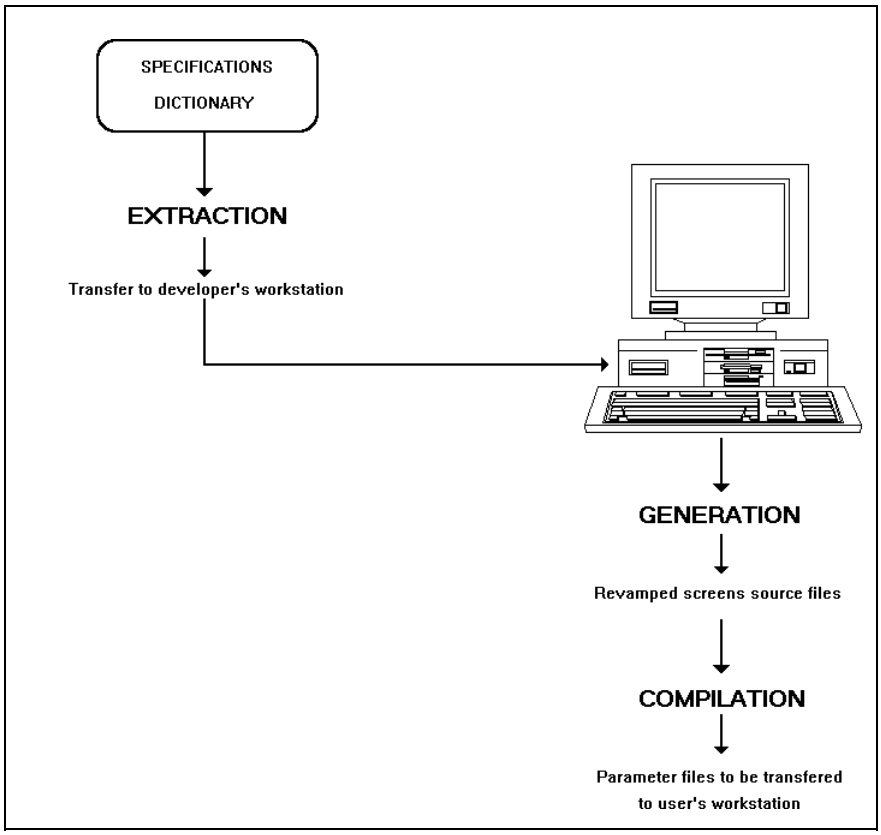
The former need to generate all the files required by the revamping of their applications, before installing them on the user workstations. They should therefore read the first five chapters.

The latter are supplied with the revamping files and therefore do not need to go through the generation process. They should read chapter "Examples of PAW revamping", subchapter "Revamping an IBM product: DSMS" which gives them specific information and, when necessary, specifies which parts of the other chapters they should read.

1. Presentation of the Developer's and User's Workstations

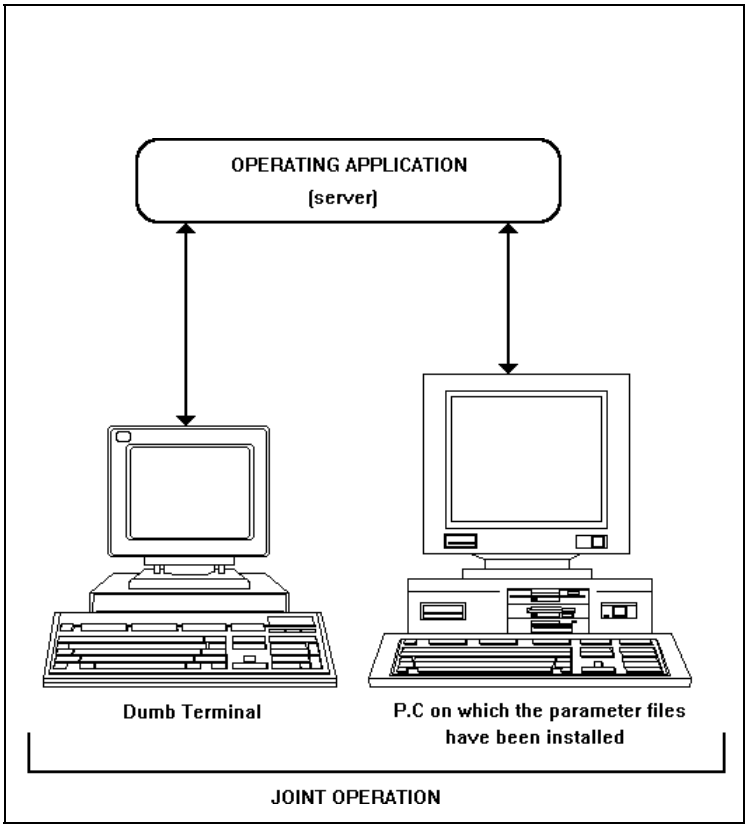
1.1. Developer's Workstation

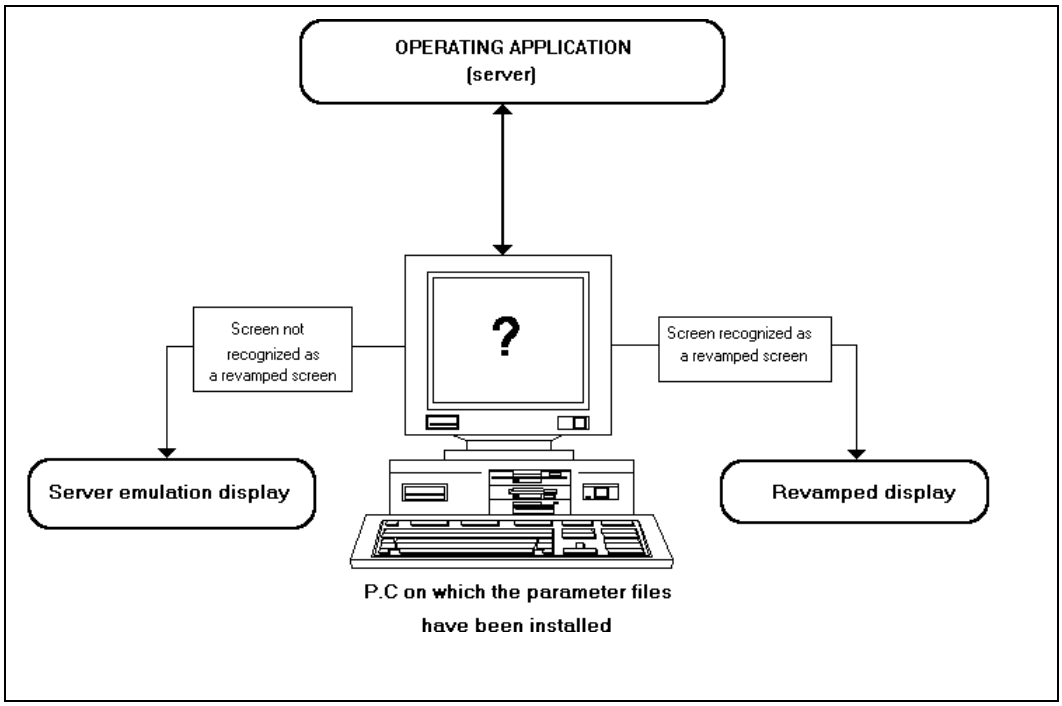
The PAWGEN module generates the revamping parameters files for applications developed with the PACBASE DIALOGUE module on the basis of information extracted from the Database. This generation step is automated and requires very little intervention from the developer.



For hardware and software requirements, see the Product Brief or your technical support.

1.2. User's Workstation





The user's workstation is a personal computer on which the developer has installed PAW and transferred the revamping parameter files compiled on his/her workstation.

Existing applications are not in any way modified: only the user interface is new.

PAW works in "mixed" mode: the workstation uses the emulation mode whenever it receives a screen that has been modified since the revamping was done, or for which there is no local revamping.

2. Installation

The first step includes installing the developer's workstation as it is from this that you can revamp the screen before transferring them to the user's workstation.

After installing the developer's workstation, you can install the user's workstation and then proceed to revamp the application or vice versa.

2.1. Installation Procedure

To install a developer's or user's workstation, run WINDOWS and select the File menu, then 'Run'. You can choose to install PAW:

- from the diskettes. For this, insert the first diskette in the disk drive (A or B), select this drive (by typing A: or B:) followed by typing INSTALL.
- from the server where the contents of the two installation diskettes are in two sub-directories. Activate the INSTALL program by indicating it's access path: e.g.: X:*disk_path*\DSK01\INSTALL, where X = server unit where the disk contents were copied, DSK01 = first sub-directory.

The procedure displays a series of dialogue boxes through which you can define your work environment.

2.1.1. Common Characteristics of Dialogue Boxes

All the dialogue boxes contain two push buttons:

- an OK button which passes to the next dialogue box recording your choices,
- a Cancel button which interrupts the installation. If you click on 'Yes' in the message box displayed you quit the installation procedure, and all the choices selected to that point are lost. If you click on 'No' you go back to the point where you originally clicked on Cancel.

To the right of the message and dialogue boxes is the EXIT icon, used to exit the installation procedure; it cancels all the choices selected to that point. This icon has the same function as the Cancel button, but unlike the Cancel button it is always available.

2.1.2. 'Installation Language' Box

You must choose between French (default language) and English, for the language used in the installation procedure's dialogue boxes and in the PAWGEN execution report.

2.1.3. 'PAW user language' box

You must choose between French (default language), English and Spanish for the language used in the error messages displayed in the revamped application.

2.1.4. 'ENVIRON.PRM Directory' Box

You must type the executables complete path. The '**c:\paw\exe**' default is modifiable.

If the directory already exists (if you have re-installed PAW), the procedure will search it for a possible ENVIRON.PRM file which contains the environment parameters. If the file is there the installation procedure proposes, by default, choices corresponding to the existing ENVIRON.PRM file. You only have to validate these choices.

The start of the path will be the default root directory proposed in the following dialogue box.

2.1.5. 'PAW Installation Parameters' Box

2.1.5.1. 'Workstation Type'

For 'Workstation type', indicate whether you are installing a developer or user workstation.

2.1.5.2. 'Learning Mode'

The learning and recognition of screens are based on algorithms which aim to associate a DLL with a message from the server.

By default, the recognition is based on the following criteria:

- protection or no protection of certain fields,
- values of certain fields,
- number of input fields, and the sum of their lengths,
- identity of the contents of the fixed labels of the message received by the server, and of the DLL fixed labels.

With these criteria, the revamping of an application dynamically modifying the protected/inputted argument of one or more fields is successful for a non-recognition of the modified screen. To get around this problem there are several learning modes available to the developer.

By default, this parameter is positioned at '1: standard', corresponding to the case outlined above.

Positioned at '2: by labels', this parameter is solely based on the learning and recognition of the screens on the identity of the contents of the fixed labels of the server, and of the DLL fixed labels.

Positioned at '3: customized', this parameter takes the default recognition criteria, modulating the protection criteria by taking a parameter (VARPRO) into account. This parameter indicates which fields can go from being modifiable to protected, also allowing the recognition of dynamically modified screens. For the details on this parameter, refer to chapter "Advanced Functions", sub-chapter "External Value Lists", section "To put the characteristics of a data element into contact and to enrich them".

Here is some advice allowing you to choose between the mode 1, 2 and 3:

- if the application server does not dynamically modify the attributes or if it only modifies a few fields, the standard learning mode is advised. In the second case, all the screen variables concerned by the dynamic modification of attributes should be described in the server database.

- if the application server dynamically modifies the attributes or several fields and if the screens concerned contain fixed labels, mode 2 is advised.

- if the server application dynamically modifies the attributes of several fields and if the screens concerned contain few or no fixed labels, mode 3 is advised. In this last case, do not forget to correctly update the correspondence file to indicate which fields are likely to be modified.

Positioned at '4: without learning', this parameter indicates that no previous learning is required. The recognition and the revamping of screens is done thanks to the location of a fixed label in this screen, previously seen by the user in PACBASE, and whose coordinates are memorized in line 037 of the ENVIRON.PRM file. This label, made up of a maximum of 8 characters, gives the name of the screen DLL to be revamped. Here is an example of the location of such a label:

[,L01,C001,N8,]

Here its location is given as being in line 1 (**L01**), column 1 (**C001**), and it is 8 characters long (**N8**).

If the label found in the screen is, for example, **PB16001O**, PAW will look for a DLL called **PB16001O.DLL**.

It is possible that the user has several DLL games, of different names, for the same screens to be revamped. In this case, it will define (in the ENVIRON.PRM file) character strings to concatenate at the beginning and end of the string in these screens. This last string must therefore be less than 8 characters long.

Example :

[PB,L02,C072,N4,MA]

Here, the string to identify is located in line 2 (**L02**), column 72 (**C072**), and is 4 characters long (**N4**).

If the label found in the screen is **LO10**, PAW will search for a DLL called **PBLO10MA.DLL**. If, in the following screen of the application, the string read is LO2**0**, PAW will search for the DLL called **PBLO20MA.DLL**.

The set *string_before+string_read+string_after* should never be more than 8 characters long.

2.1.5.3. 'Root Directory'

This information is not compulsory. It represents a data entry facility. If you are entering a root directory, indicating the absolute path (C:\PAW for example), it will automatically make up the start of the path of other directories. However, if you do not want all the directories to be dependant on this root directory you can modify the path of certain directories.

If you do not enter a root directory, you must state the complete path for each directory.

The default value proposed for the root directory is the first part of the executables' path indicated in the 'ENVIRON.PRM Directory' box.

2.1.5.4. 'Unit'

You must choose, from all the values proposed, the disk on which PAW will be installed.

2.1.5.5. 'Disk Space'

The information contained here is not keyable. It changes according to the unit chosen.

The total disk space, and the total disk space available before PAW installation are indicated, in bytes. The total disk space (same as the first line) and the disk space available after PAW installation are indicated.

PAW 1.6 requires approximately 1.4 Mb. The revamped DSMS takes approximately 8 Mb.

You must make sure that there is enough disk space available for all of the files already on the disk and for those you are going to install. If this is not the case, quit the installation and click on the Cancel push-button or the EXIT icon and free some disk space.

2.1.6. 'PAW Component Directories' Box

All the files necessary to run PAW are installed in these directories.

The root directory is not keyable. The value displayed (if there is one), comes from the previous dialog box.

By default all the proposed directories start by the root directory. But, you can indicate a completely different path for certain directories..

All the values proposed in the input fields are default values which you can modify.

The seven directories are:

- **Executables**: directory of .EXE files, parameter files for the generation, ENVIRON.PRM file and PAWTEST1.DAT test file. It will be called PATH-EXE.
- **Communication**: directory of .PRM and .TAB files necessary for different communication protocols.
- **Customization**: issued from installation, this directory contains the keyboard software icons' DLL. Issued from the first installation, it also contains the preference files (color, etc.) and the keyboard parameterization.
- **Revamping DLL**: issued from installation this directory contains the help in French, English and Spanish (Refer to the section "End of installation", paragraph "User's workstation" for an explication on the protection of these files). Once the revamping application is installed, it contains also contains the compiled parameterization files necessary to run the application on the user's workstation.
- **External lists DLL**: issued from installation, this directory is empty. It will be called PATH-LIST hereafter.

- **Developer's base**: issued from installation, this directory contains the description of the ClickPad's icons and a utility to regenerate the ClickPad. Once the revamped application is installed, this directory is the root of the tree is which the application DLL and HLP files are produced. It will be called PATH-BASE hereafter. This directory remains empty on a user's workstation.
- **Scripts**: only on WINDOWS. Scripts files directory and the associated files (CONNEX.PRM...).

The << button on the bottom left allows the user to return to the previous dialogue box to correct, for example, the root directory name.

2.1.7. 'Windows Directory' Box

At the end of installation a PAW icon is created in the folder of the same name. It is therefore necessary that the user gives the installer the name of the Windows directory where PAW must figure.

The directory presented '**c:\windows**' by default, but it is possible to change it to the name that you have chosen on your workstation.

2.1.8. 'Communication Protocol' Box

Here you choose the communication protocol under which the server works (e.g.: IBM/BULL 3270).

2.1.9. 'Protocol Variant' Box

Here you choose a type of communication, in the case where the protocol authorizes several.

The << button at the bottom left allows you to go back to the previous dialog box, to correct the communication protocol.

2.1.10. 'Reinstalling PAW on a Previous Version' Box

If the root directory defined at the start of installation already exists on the workstation, and only in this case, will you be asked to choose between three options:

- either recreate the directory (and its sub-directories): the previous version of PAW will therefore be deleted;

- or install the new versions above the previous one: the pre-existing tree is kept where the PAW components are installed, preserving the other possible files which are also there;

- or return to the installation steps to change the directory.

2.1.11. End of Installation

The installation procedure starts to install the sub-directories, you will be asked to insert the second diskette and to give a name for the PAW window in WINDOWS. This name will appear in the group which will include the PAW icon.

Once the installation is done, an 'End of installation' box is displayed.

2.1.11.1. Developer's Workstation

The installation procedure installs on the developer's workstation the different elements necessary for revamping the application. These elements are:

- a directory (designated by PATH-EXE) containing the PAWGEN modules (which generate the sources of the screens for revamping) and PAWLIS (optionally used to generate the lists of external values) and their parameters files, as well as the PAW module for the tests.
- a directory (designated by PATH-BASE) which is used as the root to sets of products by PAWGEN and PAWLIS.

2.1.11.2. User's Workstation

At the time of PAW installation on the user's workstation, three help files are automatically copied in the directory linked to the PATH-SCREENS parameter. These files are: PW_HLPEN.HLP (help in English), PW_HLPES.HLP (help in Spanish) and PW_HLPFR.HLP (help in French). As the revamping files used on the developer's workstation are also in this directory and as these files must be updated from time to time, these three help files must be protected (deletion of old versions and copy of the new ones). It is necessary to be able to protect these three help files (or one of them if you have only kept one language) from any accidental deletion.

For example, to protect the French help file from being deleted or altered, use the command:

ATTRIB +R PW_HLPFR.HLP

To cancel the protection, use the command:

ATTRIB -R PW_HLPFR.HLP

2.2. Customization of the Workstation

2.2.1. ENVIRON.PRM file

The ENVIRON.PRM file is created by the installation procedure in a directory whose access path is PATH-EXE. The developer can modify this file as necessary, using a text editor. However, the following rules must be respected:

ENVIRON.PRM lines have the following structure:

xxx comments [**parameter**]

where

- **xxx** is the number occupying the first 3 characters of the line.
- **comments** is optional comments.
- **parameter** is the parameter value entered between brackets.

Since lines which do not respect this structure are ignored, it is possible to introduce comment lines in the file.

The lines with numbers less than 20 are compulsory. The other lines (except line 27) correspond to an optional parameterization whose absence will not hinder the correct working of PAW.

001	Release and Language codes	[1.6 FR FR]
002	Communication Boards	[GSDLL32 GSPC32W]
003	Path of EXE files	[c:\paw160\exe]
004	User Path	[c:\paw160\ecr]
005	Path of external value lists	[c:\paw160\lis]
006	Developer path	[c:\paw160\dev]
007	Target system environment	[DOS]
020	Polling rate	[700]
021	DIALOGUE Tolerances	[0]
022	DLLs with external names	[0]
023	Learning Mode	[1]
024	User Type	[d]
027	GSCOM utilities path	[c:\paw160\com]
028	Parameters files	[c:\paw160\per]
029	Icons DLL	[ICOPAW]
030	Scripts paths	[c:\paw160\scr]
031	Display scripts window	[0]
032	DDE application name	[PAW]
033	Revamped application name	[PAW]
037	DLL identification	[,L01,C001,N8,]

- **Release and Language codes (001):** this code is made up of a release number, followed by two 2-character language codes. The first language code represents the user language and the second that of the installation. Both have been selected in the first two dialogue boxes in the installation procedure. The possible values are FR for French, EN for English and ES for Spanish.

- **Communication boards (002):** this is the name of the communication manager (the .EXE extension is implied), followed by the name of its associated parameter file (the .PRM extension is implied).

- **Path of EXE files (003):** this is the directory in which the PAW specific programs (PAW.EXE, PAWGEN.EXE, and PAWLIS.EXE) are copied. The developer needs these three programs to revamp the screens. However, only the PAW.EXE file is on the user's workstation. This parameter is the PATH-EXE of the installation procedure.

- **User path (004):** this is the compiled parameter file directory path. It corresponds to the SCREEN-PATH of the installation procedure.

- **Path of value lists (005):** this is the directory path of the compiled files containing the external value lists. It corresponds to the LIST-PATH of the installation procedure, which MUST be different from SCREEN-PATH**.**

- **Developer path (006):** this is the beginning of the directory for the files produced during the generation and compilation steps. The complete path contains the SESS (session number) and LIB (library containing the processed screens). This parameter is the BASE-PATH of the installation procedure.

- **Target system (007):** DOS system on which the revamped application will work. It conditions the PAWGEN generation of revamping files in MS-WINDOWS format. It is the system parameter of the installation procedure.

- **Polling rate (020):** this optional parameter specifies, in milliseconds, the time necessary to check the transmission between two calls of the communication board. This value should be higher than 20 milliseconds. If the value is 0, the board is not checked.

- **DIALOGUE Tolerances (021):** 0 or 1 (optional parameter). The value 0 generates a parameterization which conforms to the PACBASE description lines of the screens.
  With the value 1, PAWGEN carries out the same adjustments as PACBASE's Dialogue function.
  When a screen has a field starting on line 1, column 1, PACBASE's Dialogue function moves this field to line 1, column 2 in order to be able to insert an attribute. When a screen has a field finishing on the last column of the last line while the first field of the next line starts in column 1, PACBASE's Dialogue automatically moves the second to insert an attribute. With the value 0, there are risks of a break in contact between the local description and the grill generated by the server. In case of display problems refer the chapter "Error Management", sub-chapter "Operation Errors".

- **DLLs with external names (022):** 0 or 1. The value 0 (default) provokes the generation of parameters files with their corresponding screens as the root.
  With the value 1, the external code (MAP) is used as the root, which allows to put all the parameters files in several languages corresponding to the same screen server into the same directory (the MAPS do not have the same name from one language to the next). This parameter is compulsory and must be positioned at 1 when porting a MICROFOCUS application onto a PAW-revamped application.

- **Learning mode (023):** 1, 2**,** 3 4. This parameter specifies the type of criteria used by the Screen Recognition Algorithm. Selecting a learning mode is useful when the input field attributes have been dynamically modified by certain programs and the modified screens are no longer recognized. For more details on the use of this parameter, see sub-chapter "Installation procedure", section "PAW installation parameters", paragraph "Learning mode".

- **User type (024):** D or U. This line indicates whether the workstation works as a developer's workstation or a user's workstation.

- **Utilities path GSCOM (027):** this directory stores the communication files.

- **Parameters files (028):** this directory stores the preferences and parameters for the keyboard carried out during customization.

- **Icons' DLL (029):** this code is the name of the DLL containing the icons used in the logical keyboard.

- **Scripts' path (030):** this line corresponds to the path of the directory where the scripts files necessary for a program's execution are stored.

- **Display scripts window (031):** 0 or 1. This line displays the execution steps of the script. .

- **DDE application name (032):** by default, the value is ‘PAW’. It is possible to modify it if needs be.

- **Revamped application name (033) :** by default, the value is ‘PAW’. It is possible to modify it if needs be.

- **Revamped application release (036):** memorized in SCREENS.LRN. It is displayed in the PAW help.

- **DLL identification DLL (037):** a string of characters determining the name of the revamping DLL (learning mode 4).

2.2.2. PAWLIB.PRM file

The installation procedure creates the PAWLIB.PRM file in the directory whose access path is PATH-EXE. This file's contents are used to translate in the user's own language some of the labels that will be displayed in his/her station's ACTION and SCREEN BRANCHING menus, as well as some general labels related to the Help function.

The following table describes the installation version of the PAWLIB.PRM file.

PAWLIB.PRM lines have the following structure:

xxx comments [**label**]

where

- **xxx** is a number occupying the first three characters of the line.
- **comments** is an optional comment.
- **label** contains the label that will be used. It must be entered between brackets and its length must not exceed 36 characters.

Since the lines which do not respect this structure are ignored, it is possible to introduce comment lines in the file.

010	CMVT__	[No update]
011	CMVT__C	[Create]
012	CMVT__M	[Modify]
013	CMVT__A	[Delete]
014	CMVT__X	[Create or Modify]
015	CMVT__Y	[Transaction Code 5]
016	CMVT__Z	[Transaction Code 6]
020	OPER__P	[Redisplay]
021	OPER__A	[Inquiry]
022	OPER__S	[Next Screen]
023	OPER__M	[Update]
024	OPER__O	[New Screen]
025	OPER__E	[End]
040		[No extended help available for this screen]
041		[Help for screen:]

- 10 to 16:
  descriptive labels for the authorized values of a transaction code. These labels update the ***Action*** menu.

- 20 to 25:
  descriptive labels for the authorized values of an operation code. These labels update the ***Screen Branching*** menu.

- 40 and 4: help
  - 40: label displayed whenever there is no extended help on a particular screen (i.e. the on-line help for the revamped server screen has not been defined).
  - 41: help prefix displayed at the beginning of help panels titles on the revamped screen.

2.2.3. PAWMAK.PRM file

The installation procedure creates the PAWMAK.PRM file in the directory whose access path is PATH-EXE. This file is used to specify the link edit and compile options for files produced by PAWGEN in the BASE-PATH\SESSION\LIBR directory and/or by PAWLIS in the BASE-PATH\LIST directory. This file's records are similar to this example:

The PAWMAK.MC6, PAWMAK.MC7, PAWMAK.MC8 and PAWMAK.BL4 files used to parameterize the link editing and compiling for MICROSOFT C6, C7, C8 and BORLAND C4, are copied at the installation of the developer workstation in the executable directory. PAWMAK.PRM is not installed if it already exists in the directory so that the developer may keep his options. Otherwise, PAWMAK.PRM is the same as PAWMAK.MC8.

The structure of the PAWMAK.PRM file follows the example below:

```
****   PAWMAK DOS Microsoft C 8.00 English

[COMP]

FORMAT=cl -c/ALw -Gsw -Ot -Zpe -W2 -FPa -Tc <SRCPAW>.C


[LINK]

FORMAT=link <OBJPAW>, <DLLPAW>.DLL /align:16, NUL, LIBW.LIB
LDLLCEW.LIB /nod /NOE, <DLLPAW>.DEF

FORMAT=rc <DLLPAW>.DLL


[HELP]

FORMAT=HC31<DLLPAW>.HPJ


[UTIL]

COMMAND=NMAKE
```

Each of the three steps of compilation, link edit and generation is associated with a section whose name is written in brackets ([COMP]). In each section, one or several lines describe the command syntax.

There are three reserved words:

- <DLLPAW>: code of the DLL and help files.
- <SCRPAW>: code of the source files and the .OBJ files (temporary files),
- <OBJPAW>: code of the .OBJ files in the order of the link edit.

They are replaced by the appropriate codes at generation.

3. Revamping an Application

The PAWGEN module, which processes the data extracted from the PACBASE Dictionary, generates parameter files for the revamping of applications designed with the PACBASE DIALOGUE function. This stage is automated and requires very little intervention from the developer. For each screen, the revamping step produces a set of files to be compiled.

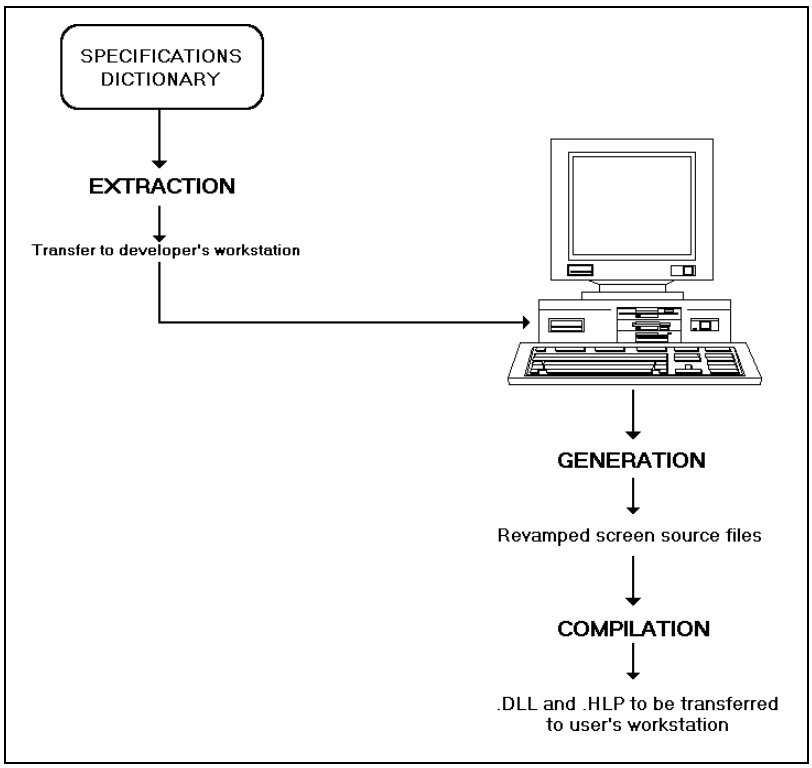
The compilation produces:

- a file containing a screen map, with the .DLL extension.
- a Help file, with the .HLP extension, containing documentary information on the screen (context sensitive Help) and its related data elements.

These files are then installed on the user's workstation.

The developer's work consists of five tasks:

- extraction, from the server, of all the data related to the dialogue whose screens are to be revamped,
- transfer of the resulting file to the developer's workstation,
- tests on the generation and compilation steps,
- generation of source files for the screens to be revamped, with the .C and .RTF (DOS MS-WINDOWS) or .IPF(OS/2-PM) extensions,
- compilation of these files,
- implementation on the user's workstation.



3.1. Extraction from the Server

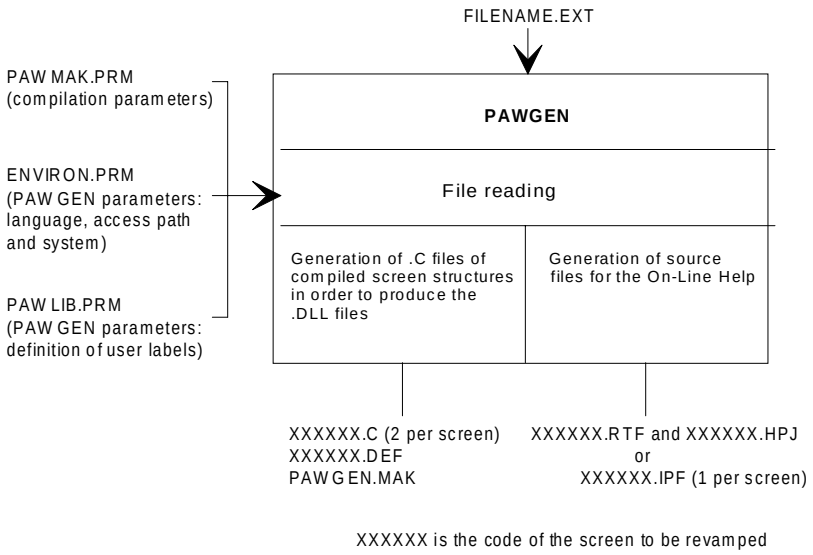
The screens that will be revamped are described in a file produced on the server by the PACBASE GPRT (GEO option C4) procedure. The procedure output is a user file: PAC7GT (or GT for the GCOS8 system) whose records are each 180-characters long (maximum).

For more information on the GPRT procedure, see the PACBASE Reference Manual and User's Manual.

3.2. Transferring the Extraction File to a PC

The downloading of the PAC7GT or GT file on a P.C. is performed by the developer. The transfer utility must be parameterized so as to keep all the special characters. The local resulting file of the transfer is FILENAME.EXT.

3.3. Generating the Screen Source Files



From FILENAME.EXT, PAWGEN obtains a set of source files to compile for each screen.

To execute this procedure, type the following command:

PAWGEN FILENAME.EXT

where FILENAME.EXT is the complete name of the file produced by the GPRT procedure and downloaded locally. This name must include sufficient indication to the access path.

PAWGEN produces .C source files and help source files .RTF for MS-WINDOWS).

These files are written in a directory whose name contains DEVELOPER-PATH\SESSION\LIBRARY where:

- DEVELOPER-PATH (line 006 in the ENVIRON.PRM file),

- SESSION is the code of the extraction session,
- LIBRARY is the code of the library from where the transaction were extracted.

In the case of the current session, the SESSION directory is always the 9999H directory. The SESSION and LIBRARY directories are created automatically from the PATH-BASE directory. Parameterization is introduced by the ENVIRON.PRM, PAWLIB.PRM and PAWMAK.PRM files studied in the chapter "Installation".

A report of the generation procedure is displayed on the screen. The developer can write it in a consultable text file and verify that it does not contain an error message. It contains:

- statistics on the running of PAWGEN;
- warning messages (warning ***).
- error messages (error *1* and error *2*).

  Error 2 signals a serious problem (system error, a parameter file missing...) which will halt processing.

A list of the errors is given in the chapter "Error Management", subchapter "Generation Errors".

3.4. Compilation

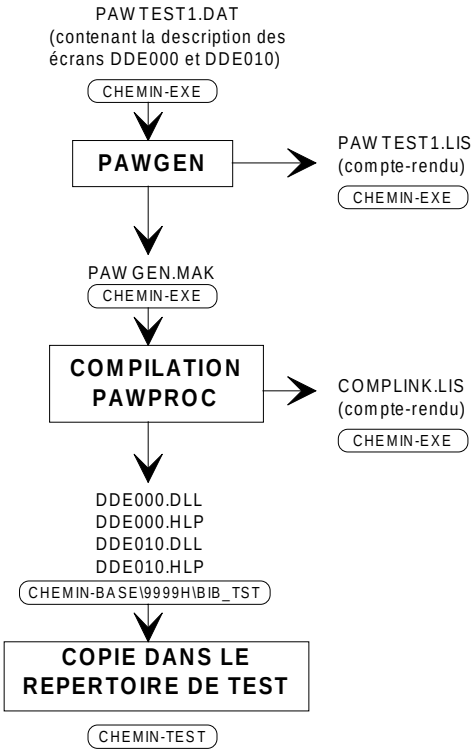
Compilations are initiated by the PAWPROC command.

The following compilations are then performed:

- compilation of .C files, followed by links to produce .DLL files;
- compilation of the Help file (.RTF) by IPF compiler or Help Compiler. The final files are written in a directory whose name is made up of the DEVELOPER PATH (line 006 in the ENVIRON.PRM file) and of the session and library names.

An execution report is displayed on the screen. Look for the "error" string to check error messages.

3.5. Tests on the Generation and Compilation Steps



The PAWTEST1.DAT file is automatically copied into the directory corresponding to the PATH-EXE parameter so that the developer can carry out tests. This file contains the descriptive of two screens: DDE000 and DDE010, which should be able to be obtained by the developer by an extraction from the central site. At the time of the test the PAWTEST1.DAT file will undergo the different phases of the revamping procedure detecting any PAW installation or parameterization error.

- Generation of parameterization

Go to the directory linked to the PATH-EXE installation parameter via the CD command.

Then execute PAWGEN.EXE using the command:

PAWGEN PAWTEST1.DAT > PAWTEST1.LIS

The generation report is written in the PAWTEST1.LIS file and can be consulted using any standard editor.

- Generation check

Print the PAWTEST1.LIS report file. Error lines begin with **error* 1* or **error* 2*. The report must not contain any of these.

- Compiling and link editing.

Enter the PAWPROC command.

If the command fails and the error message "Executable not found" **(**SYS1041**)** is displayed then the system could not find the program.

In this case, verify that:

- the program called is correctly installed on the developer's machine or on the machine's server.
- the program is either in the current directory, or in one of the directories mentioned or in the PATH environment variable.

  Use the PATH command to find out the content of this variable.

  To modify the PATH, update the environment initialization file (AUTOEXEC.BAT for DOS) and then reboot the machine.
- the names of the programs called (CL.EXE and LINK.EXE by default) are those of the compiler being used. If this is not the case, the PAWMAK.PRM files must be modified.

• Compilation and link editing check.

The PATH-BASE\9999H\LIB_TST directory must contain 2 .DLL files and 2 .HLP files. If this is not the case then there are:

- compilation errors:

  Refer to chapter "Error Management".
- link editing errors:

  There are modules libraries missing. Verify that they are installed and that they are mentioned in the environment LIBPATH variable. Use the SET command for this.

  If the environment is incorrect, update the environment files and then reboot the machine.

• Copy of the result in the test directory

The copy is done using the COPY command.

If the developer wants to first delete the old files in the directory using the DEL*.* command, he must protect the PW_HLPEN.HLP (help in English), PW_HLPES.HLP (help in Spanish) and PW_HLPFR.HLP (help in French) files depending on his choice, which are also in this directory (automatically copied during installation).

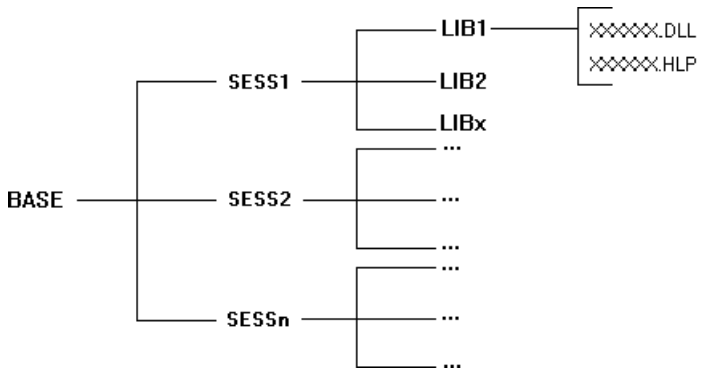
For example to protect the help file in French from being deleted or altered, use the following command:

```
ATTRIB +R PW_HLPFR.HLP
```

To remove the protection, use the command:

```
ATTRIB -R PW_HLPFR.HLP
```

3.6. Implementation



The result of the previous steps is a set of compiled files for each group of screens belonging to the SESSION\LIB directory. The exact directory name is: BASE-PATH\SESSION\LIB**.**

If the initial session is the current session, the SESSION directory is always 9999H.

In order to test the resulting parameters, the developer must copy the set of files in her/his test directory (SCREEN-PATH), by the XFERPROD command, produced by the generator.

To transfer the files to the user's workstation, the resulting parameters must be copied in the SCREEN-PATH directory on the user's workstation.

4. Error Management

4.1. Installation Errors

The installation procedure detects and points out errors resulting from the parameterization of PAW. It also detects errors resulting from a faulty parameterization of the system environment. Such errors may cause the installation process to stop.

If the system environment happens to be saturated, modify the CONFIG.SYS to allow more memory to PAW, or to deactivate resident programs that require a lot of memory. Then reboot your computer.

4.2. Generation Errors

The error messages that might be displayed following the execution of PAWGEN or PAWLIS are listed in this subchapter. Sentences in bold are the actual messages; they are followed by explanatory comments.

NUMBER	ERROR MESSAGE
DETECTED IN ENVIRON.PRM (error* 2*):**	
10001	Missing file
10002	Incorrect line
10003	Missing line
10004	Line in twice
10005	Incorrect length of parameter
10006	Incorrect path
10007	This parameter value should be 0 or 1
10008	This parameter must be a number
10009	Non-numerical version
10010	Unknown language code
10011	Unknown system code
10012	Incorrect code (ASCII or ANSI)
10013	The user path and lists' paths are the same
10014	Unknown parameter value

10043 **PAWLIB.PRM file not found or not valid**
****error* 2*** The parameter file is not in the same directory as the PAWGEN program.

10045 **PAWMAK.PRM file not found or not valid**
****error* 2*** The parameter file is not in the same directory as the PAWGEN program. .

10047 **Value list file not found or not valid**
****error* 2*** The correspondence file (**RUB-LIST.DAT** in this manual) does not exist or is inconsistent with the filename specified when generating the revamping files.

10050 **File was not processed**
The file was not processed. No parameters were produced for the revamping as a consequence of an error 10045, 10055, 10105, 10300 or 10310.

10055 **File cannot be found**
The file extracted from the PACBASE Dictionary could not be read, either because it is missing, or because the specified path is wrong.

10100 **Code not recognized**
****error* 2*** System error when revamping. Contact your technical support.

10105 **File not sorted**
****error* 2*** The file has not been sorted in ASCII. Revamping is not possible.

10110 **Sequence error**
****warning**** The record whose key is specified is not processed by PAWGEN. This message can be ignored if the revamping goes on without any serious error.

10180 **Cannot insert zone**

10185 **Cannot insert hyper**

10190 **Cannot complete paragraph**
****error* 2*** System errors when building a Help file. Contact the technical support.

10200 **Zone length = 0 in record:**
****warning**** A data element was described with a length = 0. The corresponding field will not be written in the .DLL file.

10210 **Screen is empty**
****error* 1*** The screen contains no logical or physical areas. No revamping file will be provided for this screen.

10300 **Cannot access ENVIRON.PRM file**
****error* 2*** The ENVIRON.PRM file could not be read. Check that it is not being used by another application.

20070 **Format error in file**
****error* 1*** System error when processing the screen. Related files will not be produced.

20080 **Failure while opening (mode..) File...**
****error* 1*** System error while processing the screen. Related files will not be produced. Check the file.

20400 **Field overriding another field:**
****error* 1*** Two data elements have been assigned the same field. Related files will not be produced. Make the correction in the PACBASE Dictionary.

20430 **Length of logical data element = 0. Deletion of the corresponding field. Li =.. Col =..**
****warning**** The field is deleted from the screen. The revamping files are written. To know what field is deleted, see the corresponding .C file: the logical data elements are specified with their numbers.

20440 **Field out of line. Field.. : Col =.. Len =.. Max =..**

20450 **Field out of screen. Field.. : Li =.. (Max =..) Col =.. (Max =..)**
****error* 1*** A data element is described in such a way that it extends beyond the line or screen. The screen revamping files will not be produced.

20460 **The screen's first field starts in line 1, column 1**
****warning**** If line 021 of the **ENVIRON.PRM** file is set to 1, message 20462 linked to message 20460, indicates an update on the screen description for consistency with the **PACBASE Dialog** function. Otherwise, message 20474 is linked to message 20460 and indicates a possible problem.

20462 Line 1 column 2 out-of-synch

20470 **No interval between 2 screen fields placed on 2 consecutive lines**
****warning**** If line 021 of the **ENVIRON.PRM** file is set to 1, message 20472, linked to message 20470, indicates an update on the screen description for consistency with the **PACBASE Dialog** function. Otherwise, message 20474 is linked to message 20470 and indicates a possible problem.

20472 First line shortened

20474 The screen may not be recognized

4.3. Compilation Errors

A shortage of memory may cause the compilation and link edit procedure to abend; the system then displays a message such as "Out of heap space" or "Out of far heap space".

Depending on the system and compiler (see its documentation), some options may be available for a better use of the memory (for example: /B1 C1LEXE etc...).

If problems continue, reduce the number of resident programs working during the procedure execution (modification of the CONFIG.SYS file).

Caution: some programs can be deactivated for the generation and compilation steps but must be active for the test step. Therefore do not forget to reactivate such programs when the revamping parameters are completed.

4.4. Operation Errors

It may happen that the developer expects a revamped screen and actually gets an emulated screen. On the other hand it may also happen that PAW displays a server screen using the revamping file of another, very similar, screen.

Here is a list of possible reasons, as well as some appropriate solutions.

- Screen not recognized

Check the following:

- has extraction from the host been performed?
- has it been performed with the current version of the application (the one you access via PAW)?
- has transfer to the developer's workstation been correctly made?
- is the program used for the transfer parameterized so as to keep all special characters? (very important)
- has local generation (PAWGEN) been run?
- has local generation been run on the correct file (that which has been transferred)?
- has the generation run smoothly?

- has the compilation been run?
- did it run smoothly?
- has local implementation (copy of the files in the SCREEN_PATH directory) been performed?
- are the size and date of the unrecognized screen's .DLL file found in the test directory the same as those of the original produced and compiled in the BASE\SESSION\LIB?
- are several .DLL with the same name generated? Each specific .DLL must have a specific name and can only revamp one screen. A .DLL file having the same name than another .DLL will overwrite this file which will therefore not revamp its associated screen anymore.
- the unrecognized screen contains a field that starts on line 1, column 1.

  In this case, PACBASE moves this field in line 1, column 2 to include an attribute. The generated map is therefore not in synch with its On-Line Screen Description. To solve this problem adapt the screen with the field in line 1, column 2. This has no impact on the screens for the user since the maps produced by PACBASE remain unchanged. It is also possible to introduce line 021 in the ENVIRON.PRM file so that the screen description is updated automatically.
- the unrecognized screen contains a field that ends on the last column of a line while the first field of the following line starts in column 1.

  PACBASE automatically moves the second field to include an attribute. The generated map is therefore not in synch with its On-Line Screen Description. Here also the problem may be resolved by adapting the screen description to the map produced by PACBASE, i.e. move the second field over one character.

  This has no impact at the application level as the maps produced by PACBASE remain unchanged. It is also possible to introduce line 021 in the ENVIRON.PRM file so that the screen description is updated automatically.
- the application to be revamped dynamically modifies the field attributes. In this case, check that the recognition mode is adapted to this application. See chapter "Installation", subchapter "Installation Procedure", section "PAW Installation parameters' box.

If all these issues have been checked and, if necessary, corrected, you must use the PAW diagnosis tools:

- run PAW on a developer workstation (D) on line 024 on the ENVIRON.PRM file and access the "unrecognized" screen.
- open the **Developer** menu and activate the ***Recognition*** choice. The recognition process is then activated, and a "diagnosis" displayed. The recognition is made up of three phases. Verify that the DLL corresponding to the screen is checked in phase 3. If it is the case, the message indicates the reason why it is not recognized: a gap (different label character, field gap) between the contents of the received screen (Message) and the local description of this screen (DLL). If there is a difference of characters or attributes, the check of a GSCOM trace allows you to know if the character transcoding performed at the communication is concerned or not.

Otherwise, redo the test by isolating the DLL associated with the screen in a directory in order to reduce the recognition work on phase 3. Note that a screen whose DLL is thus isolated may be recognized even though it was not during normal operation. In this case, contact your technical support.

The ***Recognition in File*** choice writes the diagnosis in the PAW.LOG file located in the customization directory.

The ***Decision Tree*** choice writes the decision tree in the executable directory. These two files and the GSCOM trace must be given to the technical support if the problem persists.

- Confusion between two screens

  The system may have difficulty in differentiating DLLs whose fields have the same positions, lengths and types (input fields or protected fields). PAW cannot distinguish between fixed labels and protected fields; therefore, if a fixed label and a protected field happen to have the same contents, the screens will be mixed up. In this case you should modify the fixed label or protected field.

- Problem of DLL loading

  If the DLL cannot be loaded, PAW sends a message with the type of error:

  - DLL not found; it does not exist in the directory. .
  - System error with system code (refer to the System's technical documentation).
  - The type of the DLL does not correspond to what was expected. PAW tried to load a DLL found in the SCREEN-PATH directory but the DLL is not what was expected. This comes from either the presence of DLL corresponding to another type (external value lists, etc.) or from the homonymy of a screen DLL with a system DLL. Therefore check homonyms. One of the DLLs used by PAW may have the same name as a system DLL (e.g. USER.DLL in WINDOWS).

5. Advanced Functions

5.1. External Value Lists

The external value lists increases the ***Help on the values*** choice available in either the ***Help*** menu of a data element or the ***Linking*** menu of a data element declared operation code (refer to section "To put the characteristics of a data element into contact and to enrich them") with values that have not come from the data elements' descriptions (-D) at the dialog level on the server (internal values), as is the default case.

If the dialogue uses external values instead of the information contained in the data elements descriptions, you can then generate - thanks to PAWGEN and PAWLIS - a version of revamping files taking these external values into account.

On the same principle, the external value lists also allow you to associate values, which are not directly described in the dialogue, to a function key. PAWGEN generates a PFKEY_ code (where _ is a blank) "function keys" field in the local description of each screen. These external values will be displayed in the ***Linking*** menu.

Two operations must be done in order for the external values to be taken into account. These operations can be done in any order but they must be finished before any testing is done:

- first operation:
  - obtaining a formatted source file, on the local drive, containing the external values. The retrieval of this values file or its creation, as well as its formatting must be done by the developer,
  - generation by PAWLIS of the external values on the local drive.
- second operation:
  - creating a parameters file to connect the data elements or the function keys with their own external values list and to indicate, for the data elements, if they are the operation codes or is they can become protected by modifying their attributes,
  - generation by PAWGEN of revamping files.

5.1.1. Structure of External Value Source Files

Records of the local source file containing the external values, called LISTVAL.DAT hereafter, must all have the following structure:

[LIST-NAME**][**VALUE-LENGTH**][**VALUE**][**VALUE-LABEL**]**

where

- LIST-NAME is the name of the external value list (this file takes the .DLL extension) in which the record must be stored. This name's length must be 8 characters (if it is shorter, completed by adding blank characters). The same LIST-NAME will appear at the beginning of all the records associated to one data element. Records having the same LIST-VALUE as a heading must be consecutive in the external value source file.

- VALUE-LENGTH specifies, with two characters, the length of the value defined in the record. This length can be any integer value between 01 and 99.

- VALUE is the value that will be displayed in the ***Help for Values*** dialog box. Its length must be that specified in the VALUE-LENGTH described above.

- VALUE-LABEL is the label associated with the external value. This label also appears in the ***Help for Values*** dialog box. Its length is free, with the restriction that the total record length must not exceed 255 characters.

Note The number of external value source files is not restricted as long as they are built according to the structure described in this paragraph.

An example of a file which supplies the external values displayed in the ***Help for Values*** dialog box:

RUBREGIO0201AIN

RUBREGIO0240LANDES

RUBCOUNTRY01FFRANCE

RUBCOUNTRY02GBGREATBRITAIN

Exemple d'un fichier qui fournira, dans le menu ***Enchaînement***, les valeurs externes associées à la touche fonction F1 :

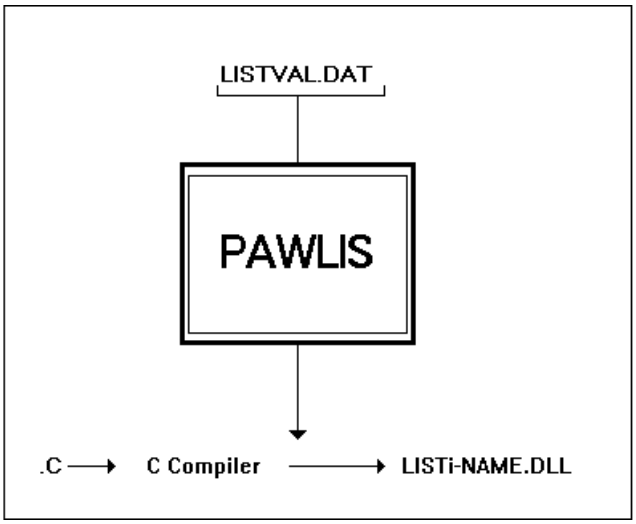
PFKEY1 0201Same Screen

PFKEY1 0200End of Converstation

PFKEY1 02A1Test

PFKEY1 0202Next

5.1.2. Local Generation of the External Value Lists



Using PAWLIS, the developer generates, from the LISTVAL.DAT file, a set of external value lists that will automatically be written in the \LIST sub-directory linked to the directory specified by the base-path parameter. The generated files have .C and .DEF extensions; they are used as entry files for the compilation.

To generate the external value lists, the developer must use PAWLIS, and specify the LISTVAL.DAT file name as a parameter, as follows:

PAWLIS LISTVAL.DAT

Once processed by PAWLIS and compiled, the above example file will produce the RUBREGIO.DLL, COUNTRY.DLL and PFKEY1.DLL files.

5.1.3. Compiling and Checking the Resulting Files

The compilation of the resulting files is triggered by a batch procedure generated by PAWLIS.EXE.

Compilations are initiated by the PAWPROC command.

The compilation of the .C files, and the link edit that follows, produce the .DLL files. These files must be copied by the developer in the directory which corresponds to the LIST-PATH of the installation procedure.

The execution report is displayed on the screen. The developer can include it in a text file for reference, after checking that it does not contain any error message (look for the "error" screen). If it contains error messages, follow the instructions contained in chapter "Error Management".

5.1.4. To put the Characteristics of a Data Element into Contact and to Enrich them.

This procedure must be done by the developer and is carried out via a file which we call here RUB-LIST.DAT. This file puts the data element or function key in contact with it's external value list and indicates if the data element is an operation code or if it can be protected when it is first inputted.

This file will also be used by PAWGEN to generate the screen's DLL, taking the external values into account. The RUB-LIST.DAT file comprises a series of recordings which must have the following structure:

[LIB**][**SESSI**][**DI**][**SCRE**][**DATAEL**][**LISTNAME**][**VAL1**][**VALU2**]**

where

- LIB is the 3-character code of the library for whose screens the external value list will be called by the Help for Values option of the Help Menu.
- SESSI is the 5-character code of the session for whose screens the external value list will be used.
- DI is the 2-character code of the dialogue for whose screens the external value list will be used.
- SCRE is the 4-character code of the screen for which the external value list will be used.
- DATAEL is the 6-character code of the data element for which the external value list will be used.
- LISTNAME is the 8-character name of the list (.DLL file) containing the external values.
- VAL1 **(**4 characters: OPER or blanks**):**
  - OPER specifies that the data element is used as operation code. Associated values are "uploaded" in the ***Branching Menu***. The presence of more than one operation code is signaled by a message.
  - Blanks specify that the data element is not used as operation code.
- VALU2 (6 characters: VARPRO or blanks) :
  - VARPRO specifies that the data element is initially an input field but that it may be protected through a dynamic modification of its attributes. If VARPRO is specified here, value 3 should be used for the learning mode. For more details, see chapter "Installation", sub-chapter "PAW Installation Parameters", section "Learning Mode".
  - Blanks invalidate the dynamic modification of attributes.

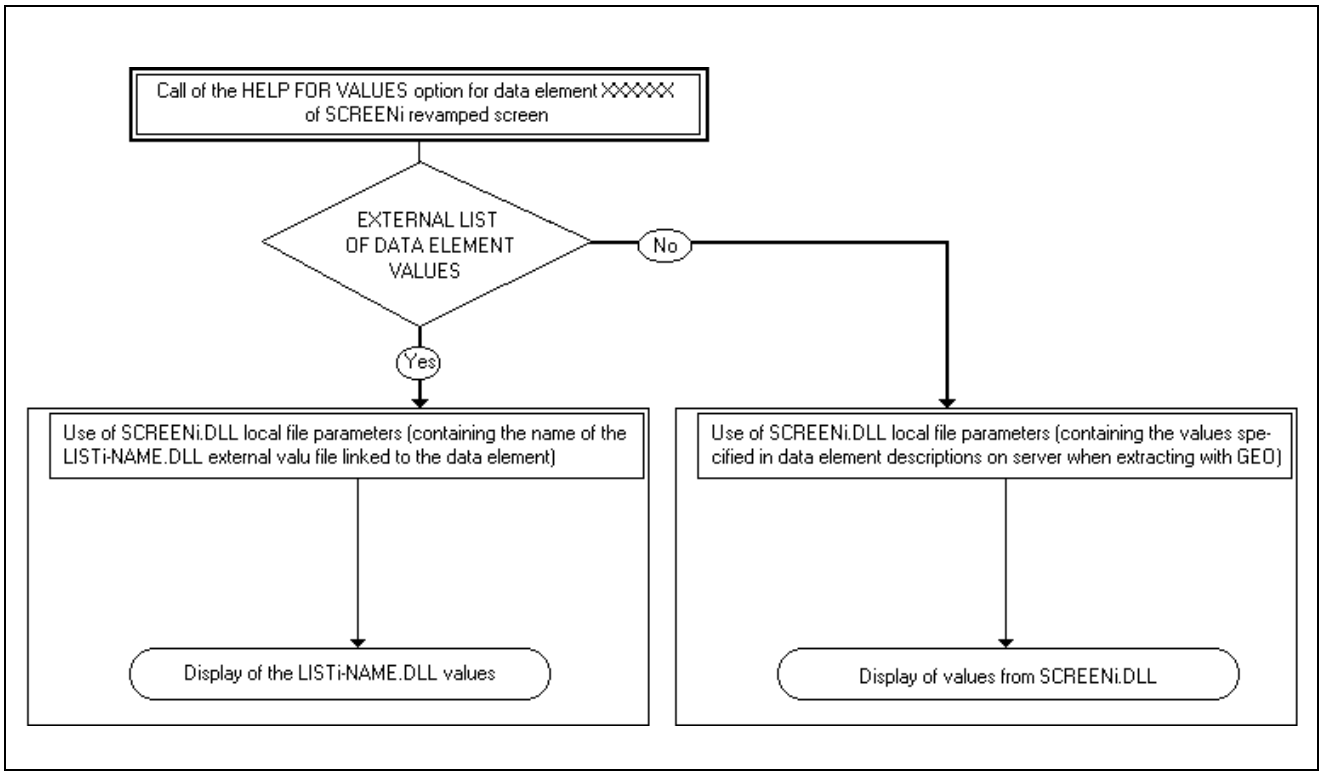
Note For each of the elements described above, use blanks to reach the length indicated.

It is possible to replace CODE-LIB, CODE-SESS, CODE-DIAL and CODE-ECR respectively with *******, *********, ****** and ********. The use of this generic character allows to respectively associate the list of external values to all the libraries, sessions, dialogues or screens.

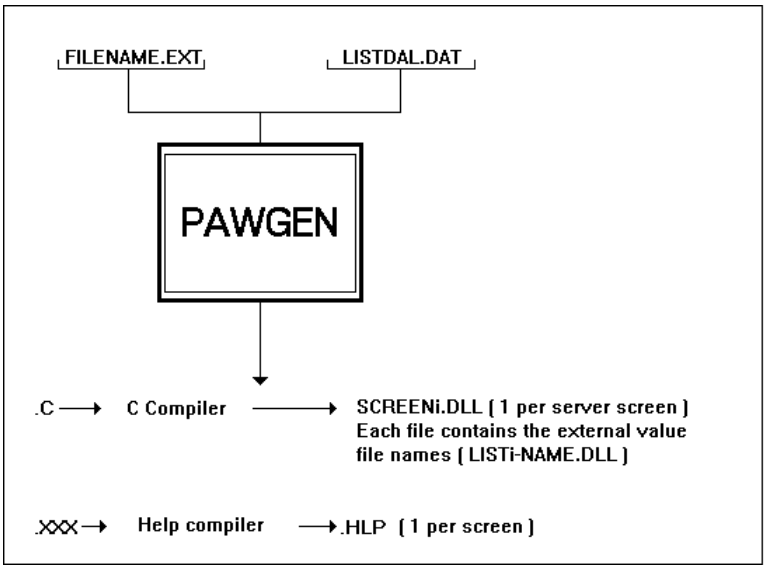
For example, the record presented below means that the values of the RUBREGIO list will be called by the REGION data element in all the D1 dialogues in any session of the L01 library. In all other cases the values described at the dialogue level will be used if they exist (see following chart).

L01*****D1****REGIONRUBREGIO

The diagram below shows in which case the external values will be taken into account following a call of the help for values option for a data element.



5.1.5. Generation of Revamping Files by PAWGEN



Once the LIST-DAL.DAT file has been finalized, the revamping process can be initiated, using the procedure described in Chapter "Revamping an application". The only difference lies in the execution command for PAWGEN: instead of typing PAWGEN FILENAME.EXT as specified in sub-chapter "Generation of Screen Source Files", type the following command:

PAWGEN FILENAME.EXE LIST-DAL.DAT

Adding LIST-DAL.DAT to the execution command affects the .DLL files of the resulting screens: instead of writing data element values in these files, PAWGEN writes references to external value lists based on information contained in LIST-DAL.DAT.

Since the files generated by PAWGEN contain only the names of the external value lists, the latter can be updated without executing PAWGEN again. Nevertheless, it is necessary to re-generate them via PAWLIS after modifying their contents.

5.2. Customizing the On-Line Help

The Help called by the ***Keys Help*** option of the ***Help*** menu is customizable and the corresponding PACBASE transactions are supplied with PAW.

When the installation is performed, the transactions associated with the ***Keys Help*** option are automatically copied in the directory corresponding to the EXE-PATH installation parameter. They are found in the 033MVT.PAC file for French release, in the 001MVT.PAC file for the English release and in the 034MVT.PAC file for the Spanish release.

The transactions supplied correspond to:

- the dialogue definition screen,
- the definition screen, the data element list (-CE) and the general documentation (-G) of the screen,
- the data element definition screens, the data element clear names being used for updating the Help index,
- the definition screen and the description of a text which contains the information of the User's Manual, split into divisions. Each division is linked to a screen data element at the general documentation level.

By default all the transactions are prefixed by "PW". This prefix can be changed using a Search/Replace function. You can also modify the Help contents by modifying the transactions themselves. The procedure described hereafter is more user-friendly: it is the modification of the transactions after their integration in the PACBASE dictionary.

The uploading of transactions to the server, and their integration to the PACBASE dictionary via the UPDT procedure are the developer's task. The developer can integrate the transactions to whichever library he/she wishes.

Note The asterisk line is not supplied; it must be inserted at the beginning of the transaction file before executing UPDT. Should it be omitted, the procedure would fail.

5.2.1. Modifying an Existing Text

Select the text you wish to modify from the clear names of the screen's data elements and general documentation, and then make your changes (modifications, translation, deletion, additions).

5.2.2. Adding One or Several Divisions to a Help Text

Creating a new Help text consists of:

- updating the screen resulting from the integration of the transactions supplied on the installation diskette, by adding to it a new data element whose clear name will be used as a Help Index input.
  The code of this new data element must have a value greater than the standard ASCII sequence codes. Furthermore, the codes ranging from PW0010 to PW0900 are reserved for IBM for possible changes. To avoid any mistake, it is strongly recommended to use a prefix with a higher ASCII value than the standard prefix (PW);

- creation of a new text division and insertion of the information to be added to the Keys Help;

- cross-reference the new text and the data element through the screen's general documentation screen.

5.2.3. Regenerating the On-Line Help

Once you have finalized the text (creation or modification), apply the revamping procedure described in chapter "Revamping an Application", to the screen resulting from the integration of Help transactions. Then perform the extraction from the server, and the transfer of the extraction file to the PC. The use or PAWGEN to generate the on-line help is slightly different to that described in chapter "Revamping an Application", subchapter "Generating the Screen Source Files". Use the following command:

PAWGEN FILENAME.EXT *

where the "*" specifies that the help is generated on the application keys. PAWGEN therefore does not supply C source but just a .IPF or .RTF extension file which must be compiled with the help compiler. The resulting file is a .HLP extension file. For more details refer to the section "Compilation and Verification of obtained files" in the subchapter "External value lists".

To test the resulting version, the developer must copy the group of files in his/her test directory (SCREEN-PATH), using the following command:

COPY PATH-BASE\SESSION\LIB\PAW_HELP.HLP SCREEN-PATH

It is necessary to rename PAW_HELP.HLP as PW_HLPEN for the English version, as PW_HLPES for the Spanish version or as PW_HLPFR for the French version.

To transfer the new version to the user's workstation, copy it in the directory corresponding to the SCREEN-PATH installation parameter of the user's workstation.

5.3. Automating the Tasks: .BAT

Most tasks in the revamping process can be automated with batch files, .BAT extension in the DOS environment. The examples shown hereafter are valid for both environments, but they should be adapted to the type of configuration in use. To adapt them to the developer's workstation, modify the directory names, the names of files to be processed, and so on. Phases preceded by "(COM)" are information that help understanding the files but are not actually part of them.

Automating tasks requires a certain knowledge of the syntax of the .BAT files, which this manual is not meant to provide. For all information on this subject, see your OS/2 or DOS documentation.

- automating the generation

```
(COM) This procedure uses 1 to 3 parameters, which are:
(COM)     2 parameters for PAWGEN.EXE.
(COM)       1 : Name of the file extracted from the host
(COM)       2 : Name of the file associating lists to data elements
(COM)           or * to generate the help on keys.
(COM)     1 parameter for PAWLIS.EXE.
(COM)       3 : Name of the external list file.

@echo off                          (COM) Deletion of echo and
cls                                (COM) clear screen.
echo     +-------------------------------------------------------------------------+
echo     |                                                                         |
echo     |    PAWG (DLL) logical screen and help file                              |
echo     |    generation procedure                                                 |
echo     |                                                                         |
echo     |    3 parameters can be used:.                                           |
echo     |        1 : Name of the file extracted from the host.                    |
echo     |            COMPULSORY.                                                  |
echo     |        2 : Name of file association between data elements and           |
echo     |            external value lists.                                        |
echo     |            or * character specifying that generation of the             |
echo     |            Key Help is required.                                        |
echo     |            OPTIONAL.                                                    |
echo     |        3 : or name of the file containing the values in                 |
echo     |            case of generation of value lists.                           |
echo     |            OPTIONAL.                                                    |
echo     |                                                                         |
echo     +-------------------------------------------------------------------------+

:PARAM                             (COM)   Test on parameters:
if p%1     == p goto FIN           (COM)   No parameters: FIN

:PGEN                              (COM)   Execution of PAWGEN.
echo .
cd REPEXE                          (COM)   Current directory is that of the .EXE files
if NOT p%2 == p goto PGEN_2

PGEN_1
echo  *** Execution of PAWGEN with ONE parameter
PAWGEN  %1     >  PAWGEN.lis       (COM) Report is written in PAWGEN.LIS
if errorlevel 1 goto STOP_GEN      (COM) If the return code of PAWGEN.EXE is
                                   (COM) other than zero the procedure is stopped.
goto COMPIL                        (COM) If not, compilation follows.

PGEN_2
echo  *** Execution of PAWGEN with TWO parameters
PAWGEN  %1 %2  >  PAWGEN.lis       (COM) Report is written in PAWGEN.LIS
if errorlevel 1 goto STOP_GEN

:COMPIL
echo .
echo  *** Construction of the .DLL and .HLP files
echo ..
call PAWPROC >> PAWGEN.lis         (COM) Report is added to the generation report

echo .
echo *** List processing ***
if p%3% == p goto CPT_RENDU        (COM) No list to be processed
cd REPEXE                          (COM) Current directory may have been changed.
PAWLIS %3
if errorlevel 1 goto STOP_LIS      (COM) In case of error, processing is stopped.

echo  *** Building the external list .DLL files ..
call PAWPROC >> PAWGEN.lis         (COM) Reprot is added to global report
goto CPT_RENDU

:STOP_GEN
echo    Error while executing PAWGEN
echo    Stop processing
```

```
goto CPT_RENDU

:STOP_LIS
echo    Error while executing PAWLIS
echo    Stop processing
goto CPT_RENDU

:CPT_RENDU
echo Report print
EDIT %1.lis                       (COM)   Replace EDIT by name of editor
goto FIN

:FIN
```

- automating the implementation

```
)
(COM) This procedure uses 2 parameters, which are:
(COM)     Library code and session number (9999H
(COM)     for the current session).

@echo off                          (COM) Deletion of echo and
cls                                (COM) clear screen.

rem No parameter -> no implementation...
if p%1 == p goto KO
if p%2 == p goto KO
goto OK

:KO
echo ...
echo Skip 2 parameters: LIBRARY and SESSION codes!
echo ...
pause
goto FIN

:OK
echo .
echo ---» Implementating %2\BIB_%1...
echo .
c:
cd PAWBAS\%2\bib_%1         (COM) Move to the directory that contains the files.
copy *.dll PAWTST           (COM) Copy of the DLL and HLP files in the test
copy *.hlp PAWTST           (COM) directory of the developer's workstation or in
                            (COM) the server directory containing the operations files.

echo .
echo *** End of implementation!!! ***
echo .

pause
goto FIN

:FIN
```

- automating the external value lists generation

```
@echo off                          (COM) Deletion of echo and
cls                                (COM) clear screen.
echo     +-------------------------------------------------------------------------+
echo     ¦                                                                         ¦
echo     ¦    PAWL procedure for generating the external value lists               ¦
echo     ¦                                                                         ¦
echo     ¦    Parameter:                                                           ¦
echo     ¦            Name of the file containing the COMPULSORY                   ¦
echo     ¦            values                                                       ¦
echo     ¦                                                                         ¦
echo     +-------------------------------------------------------------------------+

:PARAM                             (COM)   Test on parameters:
if p%1     == p goto FIN           (COM)   No parameter: END

echo *** Processing the external lists ***
cd REPEXE                          (COM)   Current directory is that of the .EXE files
PAWLIS %3
if errorlevel 1 goto STOP_LIS      (COM)   In case of error, processing is stopped
echo  *** Building the external list .DLL files ..
call PAWPROC >> PAWGEN.lis
goto CPT_RENDU

:STOP_LIS
echo    Error while executing PAWLIS
echo    Stop processing
goto CPT_RENDU

:CPT_RENDU
echo Report printing
EDIT %1.lis            (COM)   Replace EDIT with the name of the editor
goto FIN

:FIN
```

5.4. Keyboard Configuration

5.4.1. Generalities

When first running PAW, the system automatically creates a keyboard configuration file in the .EXE directory. The file's name is PAW_KBRD.PRM. It determines which key(s) will be assigned to the data transmission function. This file can be edited, and the procedure for changing this assignment or adding another data transmission keys is shown hereafter. In case of error, you can delete the file since the system will automatically re-create it in the following session.

The PAW_KBRD.PRM file contains the description of the correspondences between a key on the server and a combination of keys on the PC.

Each correspondence is coded on a line comprising:

- the coding on 18 characters. The first character indicates whether the combination is active (S) or inactive (U). Only these 180 characters are significant.
- documentary fields:
  - Server: action on the server (ex: transmit = ENT),
  - PC: PC key, sometimes with the Shift, Ctrl or Alt key.

The default transmission key is the Enter key.

Its assignment originates from the following line in the PAW_KBRD.PRM file:

coding	Server	PC	Shift	Ctrl	Alt
S002580D0000000191	**ENT**	NEWLINE			

To be able to transmit with the Ctrl key, you must activate the following line by replacing the U in the first position with an S:

Coding	Server	PC	Shift	Ctrl	Alt
U00258110000000191	**ENT**	**CONTROL**			

Coding	Server	PC	Shift	Ctrl	Alt
S00258700000105081		**F01**		**1**	
S00258720000105051		**F03**		**1**	
S00258740000105061		**F05**		**1**	
S002588C0000000191	**ENT**	**ENTER**			
S002580D0000000191	**ENT**	**NEWLINE**			
S002582C0000000171	**PA1**	**PRINTSCRN**			
S00258910000000181	**PA2**	**SCRLLOCK**			
S00258130000000131	**CLS**	**PAUSE**			
S00258001010100491	**FLC**	**e (0x65)**		**1**	
S00258700000000201	**PF01**	**F01**			
S00258710000000211	**PF02**	**F02**			
S00258720000000221	**PF03**	**F03**			
S00258730000000231	**PF04**	**F04**			
S00258740000000241	**PF05**	**F05**			
S00258750000000251	**PF06**	**F06**			
S00258760000000261	**PF07**	**F07**			
S00258770000000271	**PF08**	**F08**			
S00258780000000281	**PF09**	**F09**			
S00258790000000291	**PF10**	**F10**			
S002587A0000000301	**PF11**	**F11**			
S002587B0000000311	**PF12**	**F12**			
S002587C0000000321	**PF13**	**F13**			
S00258700001000321	**PF13**	**F01**	**1**		
S002587D0000000331	**PF14**	**F14**			
S00258710001000331	**PF14**	**F02**	**1**		
S002587E0000000341	**PF15**	**F15**			
S00258720001000341	**PF15**	**F03**	**1**		

```
S002587F0000000351      PF16    F16
S00258730001000351      PF16    F04             1
S002587F0000000361      PF17    F16
S00258740001000361      PF17    F05             1
S002587F0000000371      PF18    F16
S00258750001000371      PF18    F06             1
S002587F0000000381      PF19    F16
S00258760001000381      PF19    F07             1
S002587F0000000391      PF20    F16
S00258770001000391      PF20    F08             1
S002587F0000000401      PF21    F16
S00258780001000401      PF21    F09             1
S002587F0000000411      PF22    F16
S00258790001000411      PF22    F10             1
S002587F0000000421      PF23    F16
S002587A0001000421      PF23    F11             1
S002587F0000000431      PF24    F16
S002587B0001000431      PF24    F12             1
S002581B0000000471      REST    ESCAPE
S00258240000000012      HAUT    HOME
S00258230000000022      BAS     END
S00258090001000032      GCHE    TAB             1
S00258090000000042      DRTE    TAB
S00258240000100052      TAV     HOME                    1
S00258230000100062      TAR     END                     1
S00258080000100072      HOME    BACKSPACE               1
S002582E0000100082      EEOF    DELETE                  1
S00258080000000092      ZLIGS   BACKSPACE
S002580D0000000102      ATTR    NEWLINE
S00258250000000112      SYST    LEFT
S00258270000000122      ATTN    RIGHT
S00258250001000132      CLS     LEFT            1
S00258270001000142      ERA     RIGHT           1
S00258900000000002      NUMLOCK
S002586F0000000002      DIVIDE
S002586A0000000002      MULTIPLY
S002586D0000000002      SUBTRACT
S002586B0000000002      ADD
S002586E0000000002      DECIMAL
S00258600000000002      NUM00
S00258610000000002      NUM01
S00258620000000002      NUM02
S00258630000000002      NUM03
S00258640000000002      NUM04
S00258650000000002      NUM05
S00258660000000002      NUM06
S00258670000000002      NUM07
S00258680000000002      NUM08
S00258690000000002      NUM09
U00258110000000191      ENT     CONTROL
U00258220000002551      PAGEDOWN
U00258210000002551      PAGEUP
U00258240000002551      HOME
U00258230000002551      END
U00258260000002551      UP
```

col.	length	meaning
1	1	: S if key is active : U if key is inactive
2	5	: value of message
7	2	: virtual key (hexa)
9	3	: key code (numeric)
12	3	: booleans for Ctrl - Alt - Shift
15	3	: function identification ex: for internal functions, values from 001 to 014
18	1	: 1 if transmission to host : 2 if internal function

CTRL + F1 -> CTRL + F12	F1 … F12
CTRL + SHIFT + F1 -> CTRL + SHIFT + F12	F13 … F24

5.4.2. Local Functions

The local functions are preprogrammed functions that are activated by certain keys. The user defines these keys in the PAW_KBRD.PRM file, associating them with the desired internal function number (columns 15 to 17).

These functions are as follows:

no.	**functions**
0	does nothing
1	start of the field
2	end of the field
3	start of the previous field
4	start of the next field
5	start of the first field on the screen
6	start of the last field on the screen
7	deletion of the start of the field
8	deletion of the end of the field
9	deletion of the previous character
10	start of the first field of the next line
11	move a character to the left
12	move a character to the right
13	deselect the previous character
14	deselect the next character
15	print the screen
16	move a character to the right with automatic tabulation in the previous field
17	move a character to the right with automatic tabulation in the next field
18	recall the last local function used
19	start of the last field of the previous line
20	start of the last field of the next line

(The fields mentioned above must be understood as being keyable fields).

5.4.3. Values of Keys (to be inserted in PAW_KBRD.PRM)

These values must be inserted in columns 7-8 (virtual key).

Hex value	key name
1b	Escape
2d	Inser
24	Home
2e	Del
23	End
21	PageUp
22	PageDown
25	Left arrow
26	Up arrow
27	Right arrow
28	Down arrow
6d	Subtract
6f	Divide
6a	Multiply
6b	Add
6e	Decimal
60	num.0
61	num.1
62	num.2
63	num.3
64	num.4
65	num.5
66	num.6
67	num.7
68	num.8
69	num.9

5.5. ClickPad

5.5.1. Characteristics

The ClickPad is a window which allows the user to perform four types of actions via the mouse: standard transmit actions (simple transmit, screen clearing, restoration, print request), and actions found in the Action, Screen Branching and Scripts menus.

Each action type corresponds to a button type (standard, action, screen branching and scripts) whose display is parameterized by the user in the 'Options, Preferences' menu.

By default, these buttons are push-buttons. However you may change them into icons. Icons are stored in a DLL-type file which is found in the user parameter file directory (line 028 of the environment file) and whose name (filename without extension) is indicated on line 029 of the environment file.

When PAW is installed on a developer's workstation, the program copies onto the user directory the files which contain the icons of the standard transmit buttons, as well as a C source file of the DLL and an example procedure file (DOICOPAW.BAT) for the compiling and link editing of the icon DLLs. This file can be modified or adapted to a compiler different from the one in use (MICROSOFT C 8.00). In the link edit, use the following files: LIBENTRY.OBJ, MDLLCEW.LIB (or LDLLCEW.LIB) and LIBW.LIB.

5.5.2. Changing the Standard Icons

You can customize the icons associated with the ClickPad. These icons are stored in a DLL file (for example ICOPAW.DLL in the directory of customization files) resulting from the compiling and link editing of the following files:

ICOPAW.C: C source

ICOPAW.RC: WINDOWS resources

The ICOPAW.C file must not be modified.

The ICOPAW.RC file is made up of:

- an INCLUDE <WINDOWS.H> line which must not be modified.
- definition lines for the standard buttons which must not be modified.
- lines for the association of icons (files with ICW extension) with integers (represented by a character string, ICO_ENT for example).

5.5.2.1. Adding New Icons in the ICOPAW.RC File

- Step 1: Introduce a new line for the association of each icon, according to the following syntax:

1000 ICON NEWICON.ICW

where:

- 1000 is the integer, between 1000 and 9999, which identifies the icon.
- ICON is a keyword.
- NEWICON.ICW is the code of the icon description file. This file can be created with the help of PAINTBRUSH or SDKPAINT for example and copied in the developer's root directory: BASE-PATH.

- Step 2: Execute the DOICOPAW.BAT procedure (the name of the DLL must be input as the parameter).

Note: to be used, the DLL file must have its initial name, that is the one given at its creation by DOICOPAW.

- Step 3: Copy the created DLL file in the customization files directory:

If you do not want to overwrite the previous DLL file with the created one and if you want to use the new icon parameterization, you can choose one of the following two solutions:

- rename the previous DLL file,
- modify line 029 of the ENVIRON.PRM file with the name of the new file.

Note: You cannot associate the same number (or character string) to two different ICW files.

5.5.2.2. Changing the Icon of a Standard Button

Each standard button is referenced in the ICOPAW.RC file and associated with an integer. To associate a new icon to this button, you must:

1. Copy the icon file (ICW extension) into the developer's root directory BASE-PATH (work directory),
2. In the ICOPAW.RC file, associate the icon file to the integer which represents the specific button,
3. Run the DOICOPAW.BAT procedure file which creates the DLL; indicate the DLL filename as parameter (for example NEWICOPAW.DLL);
4. Copy the created DLL file in the directory of customization files.

5.5.2.3. Associating an Icon with an Action or Screen Branching Button

The non-standard buttons come from either the PACBASE database or from a list of external values. To associate one of these buttons with an icon, give it a numeric identifier. This identifier is a four-digit integer included between 1000 and 9999 and written between brackets at the beginning of the label of the value taken into account in the Action or Screen Branching menus. Then run the procedure which creates the DLLs (DOICOPAW.BAT) and copy the DLL in the adequate directory.

5.5.2.4. Associating an Icon with a Script

To associate an icon with a script, you have to write the four-digit integer which identifies the icon at the beginning of the script label (in the corresponding PRM extension file). Then run the procedure which creates the DLLs (DOCICOPAW.BAT) and copy the DLL in the adequate directory.

If the loading of the DLL files, or if the program cannot display the icon, the button will be displayed as a push-button with the label text.

6. Examples of PAW revamping

6.1. Porting a MICROFOCUS Application onto a Revamped Application

6.1.1. Architecture of a DOS MICROFOCUS Dialogue Application

A DOS MICROFOCUS Dialogue application is made up of the following programs (see the diagram below):

- MONITOR.EXE is the Dialogue monitor. It controls the Dialogue's screen flows. It is the first program to be executed when the application is started. The SCREEN1.EXE, SCREEN2.EXE, ... programs are specific to the Dialogue's screens.
- ZAR980.EXE is used as an interface between the screen programs and the Input/Output programs.
- SCRSAISI.EXE is the Input/Output programs which controls video display and keyboard input.

The ZAR980 and SCRSAISI Cobol source programs are supplied.

To generate these programs, your compilation command file should resemble that presented in paragraph "Examples of compilation command files".

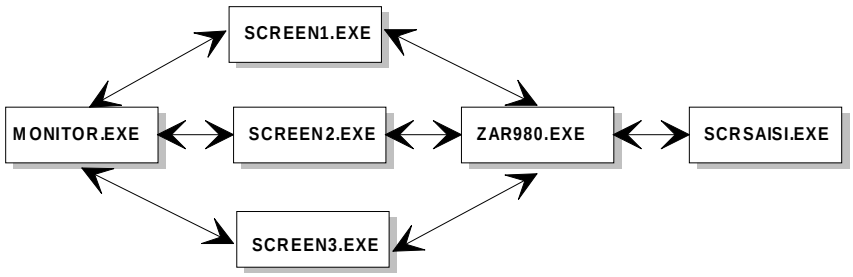


Figure 1: Structure of a DOS MICROFOCUS O.L.S.D. application

6.1.2. Architecture of the Revamped Application under WINDOWS 3

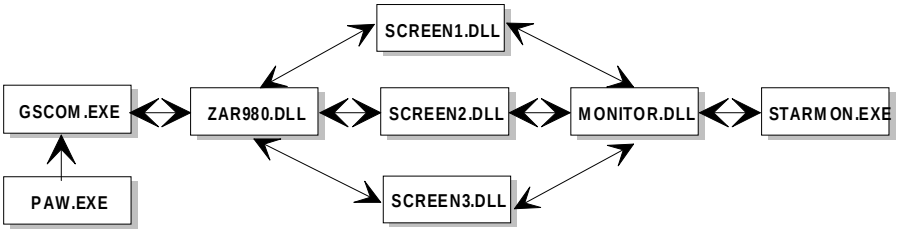
The Dialogue screen flow (Figure 2) of a MICROFOCUS application revamped under WINDOWS is similar to that of a MICROFOCUS application under DOS, with the difference that the monitor is not run directly, but through PAW and via the GSCOM.EXE and STARTMON.EXE programs.

The ZAR980.EXE and SCRSAISI.EXE programs are replaced with a unique program called ZAR980.DLL. This program uses GSCOM.EXE to control input and output (display and keyboard input) exchanges between the MICROFOCUS application and PAW.

The Cobol source code will not produce .EXE files but .DLL files (refer to the WINDOWS Software Development Kit for a complete description of the dynamic libraries concepts). The advantage of the .DLL files is that they extend the size limitations of the executables code (under DOS, the maximum size of a program is 500 Kb, whereas it may be two or three times larger in WINDOWS). Moreover, WINDOWS can load several programs of this size simultaneously.

You will find an example of production of .DLL files for the monitor and screen programs in paragraph "Examples of compilation command files". This generation is obtained from the .OBJ files generated during the COBOL compilation of the application under DOS.

The PAW, GSCOM, STARTMON and ZAR980 programs are supplied.



2. Structure of the same application revamped in WINDOWS 3

6.1.3. Notes and Recommendations

- The revamping of MICROFOCUS applications can only be done when the applications have been generated from PACBASE.

- Make sure that the monitors' source codes do not contain any "STOP-RUN" Cobol instructions. If they do, replace these instructions by the "GOBACK" instruction, using a text editor. With recent versions of PACBASE this generation is performed automatically.

- Do not use any "DISPLAY" instructions.

- In order to avoid WINDOWS conflicts between the .DLL files of the screens and monitor programs and those of the revamping application, make sure that the MAP names entered on the screens' definition screens are *different* from the program names.

  Example *program name:* ***DO0002***
  MAP *name:* ***DO0002M***

- The DLL called ZAR980.DLL is the installation DLL. Do not create a DLL of this name from the COBOL source used to produce the ZAR980 of the full screen version. Insert a line 022 in the ENVIRON.PRM parameter file (in PAW's directory), as shown hereafter:

```
--- ENVIRON.PRM ---
001 ...
...
022     external DLL file names      [1]
```

- The MICROFOCUS application's DLL directory should be included in the PATH and COBDIR environment variables.

- In a WINDOWS MICROFOCUS application, the relation between internal file names and external ones is identical to what is under DOS, i.e. it is determined by the environment variables set. Therefore, if you want to run several PAW sessions on different monitors (it is not possible to run several PAW sessions on the same monitor), you must assign a specific internal file name to each application.

  Example :
  ***APPLI1** application internal name of the **APP1LE** error message file*
  ***APPLI2** application internal name of the **APP2LE** error message file.*

  External files, however, may have identical names.

  Example :
  C:\APPLI1\LE for the error message file of application 1
  C:\APPLI2\LE for the error message file of application 2

  with the following settings for the environment variables:
  SET APP1LE=C:\APPL1\LE
  SET APP2LE=C:\APPL2\LE

  - Note that in the examples of compilation command files shown hereafter, only the MONITOR program is linked with the EXTFH.OBJ and EXTERNL.OBJ modules (which contain the COBOL routines for accessing the files provided with the MICROFOCUS WORKBENCH). The size of each screen program is therefore reduced by around 70 Kb.
  - Check that the directory containing the ZAR980.DLL program (supplied with the PAW environment) is included in the PATH variable of the AUTOEXEC.BAT file.

- Although WINDOWS uses the ANSI format, the format of the external files processed by the PAWLIS and PAWGEN generators should be ASCII.

6.1.4. Examples of Compilation Command Files

- Example of the compilation command file of a MICROFOCUS O.L.S.D. application in DOS:

```
         rem --- COBOL COMPILATION ---
cobol ZAR980.CBL ;
cobol SCRSAISI.CBL ;
cobol MONITOR.CBL ;
cobol SCREEN1.CBL ;
cobol SCREEN2.CBL ;
cobol ... other screens...

         rem --- LINK EDITING ---
link SCRSAISI.OBJ, SCRSAISI.EXE, NUL,
          /nod LCOBOL.LIB COBAPI.LIB, NUL
link ZAR980.OBJ, ZAR980.EXE, NUL,
          /nod LCOBOL.LIB COBAPI.LIB, NUL
link MONITOR.OBJ EXTFH.OBJ EXTERNL.OBJ XFHNAME.OBJ,
          MONITOR.EXE, NUL, /nod LCOBOL.LIB COBAPI.LIB, nul
link SCREEN1.OBJ , SCREEN1.EXE, NUL,
          /nod LCOBOL.LIB COBAPI.LIB, NUL
link SCREEN2.OBJ , SCREEN2.EXE, NUL,
          /nod LCOBOL.LIB COBAPI.LIB, NUL
link ... other screens...
```

- Example of compilation command file for revamped MICROFOCUS O.L.S.D. application in WINDOWS:

```
         rem --- COBOL COMPILATION ---
COBOL MONITOR.CBL ;
COBOL SCREEN1.CBL ;
COBOL SCREEN2.CBL ;
COBOL other screens ...

         rem --- LINK EDITING ---
LINK @MONITOR.LNK
LINK @SCREEN1.LNK
LINK @SCREEN2.LNK
LINK other screens ...
```

where @**MONITOR.LNK** contains the following lines:

```
MONITOR+CBLWINL+LIBINIT+EXTFH+EXTERNL+XFHNAME, MONITOR.DLL,,
LCOBOLW+LCOBOL+COBW, MONITOR.DEF /nod /noe;
```

where **MONITOR.DEF** contains the following lines:

```
LIBRARY         MONITOR
DESCRIPTION     'Monitor PAW and WINDOWS'
EXETYPE         WINDOWS 3.0
CODE            PRELOAD MOVABLE DISCARDABLE
DATA            PRELOAD SINGLE NOT SHARED
HEAPSIZE        0
EXPORTS         MONITOR @1
```

where @**SCREEN1.LNK** contains the following lines:

```
SCREEN1+CBLWINL+LIBINIT,SCREEN1.DLL,,
LCOBOLW+LCOBOL+COBW,SCREEN1.DEF /nod /noe;
```

where **SCREEN1.DEF** contains the following lines:

```
LIBRARY         SCREEN1
DESCRITPION     'Screen PAW and WINDOWS'
EXETYPE         WINDOWS 3.0
CODE            PRELOAD MOVEABLE DISCARDABLE
DATA            PRELOAD SINGLE NOT SHARED
HEAPSIZE        0
EXPORTS         SCREEN1 @1
```

The same will apply to all the screens of the dialogue.

Note The compilation commands and the libraries to be used at link edit time may vary according to the COBOL releases. You must then refer to the MICROFOCUS documentation. On the other hand, the .DEF files only depend on the WINDOWS release.

6.2. Revamping an IBM Product: DSMS

6.2.1. DSMS Revamping: Introduction

IBM has developed a complete revamping of the DSMS screens. DSMS users may thus benefit from the advantages of a PC graphic environment when using their mainframe application.

The developer's task is limited here to the installation of PAW and the revamping and help files, provided on installation, on the users' workstations.

The developer also needs to build external lists which will contain the values entered in the DSMS tables and will be called by the ***Help for Values*** option of the ***Help*** menu. To build these external lists, the developer needs to install the revamped version of DSMS on his/her own workstation. The resulting value lists should then be installed on the users' workstations, together with the revamping and help files supplied on installation. For details on the external value lists refer to Chapter "Advanced Functions", Sub-chapter "External Value Lists".

The installation of the revamped DSMS version comprises the installation of the **local word processor**. The .DLL files referring to the Language, Product and Subsidiary tables are automatically copied in the directory of external value lists (see below). If you modify the access paths to these files after the installation, you must update these access paths in the ***Advanced Preferences*** dialog box which you access through Menu ***Options***, choice ***Preferences*** of the word processor.

For a complete description of the local word processor of the revamped version of DSMS, refer to the corresponding chapter in the User's Manual.

6.2.2. Installation

The installation of the revamped DSMS is the same as for PAW, if only the 'PAW component directories' dialog bix appears, as well as the directories present in the PAW installation, a DSMS word processor program directory and a Windows directory which will receive the TT.INI file (general word processor parameterization).

For a detailed description of the installation procedure, refer to the chapter "Installation", subchapter "Installation procedure".

To finish off the installation of the developer's workstation, refer to the chapter "Installation", subchapter "Customization of the installed workstation". There you will find explanations for the following three files:

- the ENVIRON.PRM file.
  The 001, 021 and 022 parameters do not concern the developer revamping DSMS.

- the PAWMAK.PRM file.
  As regards revamping DSMS, this file indicates the compilation and link editing options for the files produced by PAWLIS.EXE in the PATH-BASE\LIST directory. The HELP value of OPER and the examples do not concern the developer revamping DSMS.

- the PAWLIB.PRM. file.
  Translation in the user language of certain labels displayed in the ***Action*** and ***Linking*** menus.

Also, a certain number of Script files of examples are in the <*root**scripts*\SAMPLES> directory.

6.2.3. Building the External Value Lists

In order for the values entered in the DSMS tables to appear when the user selects the Help for Values option of the Help menu (see the PAW User's Guide), the developer must create external lists containing these values.

The operation involves producing, on the workstation, a formatted source file containing the external values.

To extract the contents of the DSMS tables, run the DEXH procedure from the DSMS database (host). For details on this procedure, see the DSMS Operations Manual. Downloading the generated file is the developer's task. The developer may also produce the source file "manually" on the workstation, but the particular structure of the file should be repeated.

To complete the operation, refer to:

- paragraph "Structure of external value source files" in sub-chapter "External value lists", chapter "Advanced functions". For DSMS, the LIST-NAME parameter will take the following values:

VALUES IN THE DSMS TABLE:	DSMS ACCESS TO THE TABLE	CORRESPONDING LIST-NAME
Contact attributes: functions	TATF	ID99TATF
Contact attributes: special responsibilities	TATR	ID99TATR
Gravity: change	TGRC	ID99TGRC
Gravity: event	TGRE	ID99TGRE
Languages	TLA	ID99TLA
Options	TOP	ID99TOP
Phases	TPH	ID99TPH
Products	TPR	ID99TPR
Regions	TRE	ID99TRE
Status: change	TSTC	ID99TSTC
Status: event	TSTE	ID99TSTE
Status: site	TSTS	ID99TSTS
Subsidiaries	TSU	ID99TSU
Types	TTY	ID99TTY
User definitions	TUD	ID99TUD

If you used the DEXH procedure, the file will automatically have the right structure and you may proceed to the local generation step.

- paragraphs "Local generation of the external value lists" and "Compiling and checking the results files, in chapter "Advanced Functions". The files resulting from the compilation and found in the BASE-PATH\LIST directory must be copied manually in the LIST-PATH directory.

6.2.4. Keyboard Configuration

When you first use the revamping DSMS, a keyboard configuration file is automatically created. Its name is PAW_KBRD.PRM, and it is detailed in paragraph "Keyboard configuration" in chapter "Advanced Functions".

6.2.5. If problems arise

If problems arise, refer to chapter "Error Management".

7. PAW DDE Server

You can use PAW as a DDE (Dynamic Data Exchange) server of a client WINDOWS application. With PAW as a DDE server, a client WINDOWS application asks WINDOWS to open a DDE Dialogue with PAW. From the opening to the closing of the dialogue between the two programs, the client application can either ask (REQUEST) or send (POKE) data to PAW or ask for the execution of commands. In all cases, the information traveling between the applications are character strings.

You will find a short description of the principles and operation of DDE in the following pages. A good knowledge of these principles and operation is useful to get the most out of it.

7.1. Characteristics of a DDE Dialogue

To open a DDE dialogue, you have to define the *application* and the *topic* the client application will talk to. The object of the dialogue can then be either an *item* or a *command.* These four terms are described below:

- the *application* corresponds to the name of the server (default value: PAW); this name is defined by the user and saved in the ENVIRON.PRM file on line 032;

- the *topic* is the radical of the used connection script filename and thus corresponds to the server application which the client application will talk to; PAW authorizes also as *topic* the use of the character string SYSTEM (DDE standard), which allows you to open DDE dialogues independently from the current server application;

- an *item* corresponds to the data asked or sent by the client application to the server application; this data can either be a display field of the screen or the entire screen. A field is identified either by its coordinates, or logically as in the script language. This identification can be completed by the screen code (optional);

- a *command* corresponds to a character string sent by the client application to the server application; this character string initiates an action: for example the display of a window, a screen branching or a transmission.

To open a dialogue with the DDE server, the client application uses the message WM_DDE_INITIATE associated to the character strings which correspond to the *application* and to the *topic.* The subject of the dialogue can be either the request or the sending of an *item* (see subchapter "Syntax of the items"), or the sending of a *command* (see subchapter "Syntax of the commands").

For more details on the sending of a message and on the opening of DDE dialogues, refer to your WINDOWS technical documentation.

7.2. Characteristics of DDE Connections

There are three types of DDE connections: *passive*, *automatic* and *active connections*. In the current version of PAW, only the passive connections are operational. In a *passive connection*, the client application asks the server for data by sending a WM_DDE_REQUEST message. The server answers with a WM_DDE_DATA. Unlike the automatic and active connections, when the data is updated on the server, it is not returned to the client which has therefore to ask explicitly for it.

SEQUENCES OF A PASSIVE CONNECTION

The three main sequences of a DDE passive connection are the *opening of the dialogue*, the *request and receipt of the value* and the *closing of the dialogue.* The conversation is made up of the following steps:

Opening of the dialogue

Client	**Message**	**Server**
Asks for the opening	WM_DDE_INITIATE →	
	WM_DDE_ACK ←	Accepts

Request and receipt of a data

Client	**Message**	**Server**
Requests a data	WM_DDE_REQUEST →	
	WM_DDE_DATA ←	Sends the data
receives the data (optional)	WM_DDE_ACK →	

Closing of the dialogue

Client	**Message**	**Server**
Closes the dialogue	WM_DDE_TERMINATE →	
	WM_DDE_TERMINATE ←	Closes the dialogue

7.3. Syntax of the Items

An item is designated by an expression made up of two groups of elements:

- in the first group, the elements (separated by full-stops) indicate the type and, if necessary, the screen and the code of the corresponding data,
- in the second group, the elements (separated by commas and put between brackets) indicate the position of the data.

Here are the available items:

szScreen.FIELD [iLine ,iColumn ,iLength]	Field belonging to the szScreen and beginning at position iLine-iColumn.
szScreen.DE.szCodRub.szType [iPosition]	Data element (DE) belonging to the szScreen, of code szDeCode, of type szType (cf. types of data elements in the scripts) and of position iPosition.
szScreen.ACTIONCODE [iPosition]	Action code data element with iPosition different than 0 or 1 if necessary.
szEcran.OPERCODE	Operation code data element.
szEcran.ERRORCODE [iPosition]	Error code data element with iPosition different than 0 or 1 if necessary. This data element can not be
used	used with the WM_DDE_POKE message.
szEcran.SCREEN	Returns the whole screen.
szEcran.SCREENID	Screen code.
szEcran.GETCURPOS	Returns the coordinates of the cursor's position: [iLine, iColumn].

In all the examples above, szScreen is optional. It allows to add an additional control on the screen code.

7.4. Syntax of the Commands

A command is made up of a verb, which can be associated with a screen code and an operand.

For example, szScreen.SEND ("PF01") asks for the transmission, on the given screen, of the function key.

The available commands are:

szScreen.SCREENBRANCH(szOper)	Request of screen branching. szScreen is the code of source screen. szOper is the value of the operation code.
szScreen.SEND(szKey)	Transmission with the mnemonic code of the key (cf. the scripts) in szKey.
szScreen.SCRIPT(szCodFic)	Script execution request.
szScreen.SHOW(szShow)	Displays **PAW** window. The szShow parameter has the same values than the parameter corresponding to the SCRSHOW scripts.
szScreen.STATE(szState)	Activation or deactivation command of the **PAW** window. The values of szState are NORMAL or SILENT. In SILENT mode, PAW executes all the tasks of the NORMAL mode but either sends error codes nor execute automatic screen scripts. These scripts can be initiated at any time with the script order.
szScreen.HELP(szHelp)	Displays the **WINDOWS** help window. Depending on the value of szHelp, you get: H_ONPAW: Help on PAW; H_INDEX: Help index; H_ONHELP: Help on help; *DeCode*: Help on the Data element *DeCode;* *Blank*: Current screen help (extended help).
szScreen.SETCURPOS (iLine ,iColumn)	Positions the cursor at the defined coordinates (iLine , Column).

7.5. VisualBasic Example of PAW used as a DDE Server

Examples of VisualBasic code are shown in italics. Text1 is the name of the "edit" window of the VisualBasic application which is the DDE communication support.

1. Opening of a DDE dialogue

```
ConstMANUAL = 2
Dim String, Item, Screen

Text1.LinkTimeout = 1000
If (Text1.LinkMode = 0) Then
  Text1.LinkTimeout = 200
  Text1.LinkTopic = "PAW\NPSF"
```

Opening of a dialogue with the PAW application on the NPSF topic (root of the connection script name)

```
  Text1.LinkMode = MANUAL
End if
```

2. Repetitive field reading of the current screen (no screen code)

```
Screen = ""
For I = 1 to 18
```

Data salvage loop of the repetitive data element IDATM. The string sent at each request is of the following form: **.DE.IDATM.PRPTEC[i]** where i is the position in the repetitive.

```
  Text1.LinkItem = Screen & ".DE.IDATM.PROTEC["&Str(I) & "]"
  Text1.LinkRequest

  If (Text1.Text = "   ")Then
        Exit For
  End if
```

3. Sending a value, entered in the client application (edit box Text2), in the operation field of the current screen..

Text1.Text = Text2.Text	Reading of the code to be sent
Text1.LinkItem = ".OPERCODE"	The target data element in PAW is the operation code.
Text1.LinkPoke	Sending the data.
Text1.LinkExecute (".SEND (ENT)")	After the loading, sending request.

4 - Sending data with a check on the screen code

Text1.Text = Text2.Text
Text1.LinkItem = "ID00E2.DE.ICHOIC.OPER[1]"

Loading the ICHOIC data element (operation code) in the ID00E2 screen. Sending is only performed if the current screen is ID00E2.

Text1.LinkPoke

Text1.LinkExecute (".ID00E2.SEND (ENT) ")

Sending if the current screen is ID00E2.

8. Script Language

8.1. Introduction

The objective of the script files is to write procedures that would otherwise be entered manually. This is done by a series of instructions, designed by you, and sent transparently to the host site.

Procedures to host sites must be extremely precise. Therefore, we have provided an interpreted language where the control structures are more plentiful than those available through batch.

Examples of scripts and of a CONNEX.PRM file are automatically copied in the SAMPLES subdirectory at the PAW installation.

8.2. Types of Scripts

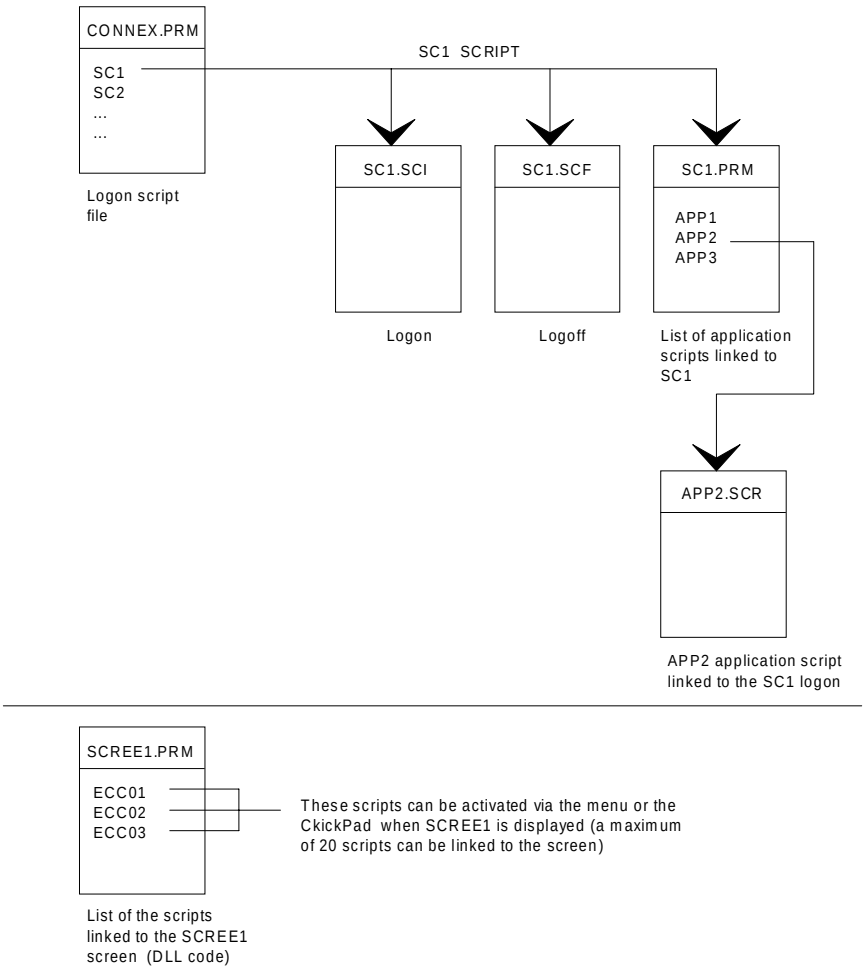
Scripts belong to two categories:

- Logon scripts (with a SCI extension) and logoff scripts (with a SCF extension).
  These scripts perform an automatic logon or logoff.

- Application and screen scripts (with a SCR extension).
  These scripts perform sequences of instructions during the work session, i.e. between the logon and logoff.
  The application scripts can be activated at any time, either via the Script menu, or via the ClickPad.
  The screen scripts are displayed (and can be activated) only when the screen to which they are linked is displayed.

NOTES:
The maximum number of scripts that can be activated for a screen is 50.
The PAW installation gives you examples of logon (logoff) scripts. These scripts are automatically copied in the SAMPLES subdirectory.

The chart overleaf shows the different types of scripts.



8.3. Implementation of Scripts

8.3.1. Parameters

All the files required for the use of PAW scripts are grouped in a specific directory (line 030 of the ENVIRON.PRM file).

8.3.1.1. Line Structure of the Scripts

The lists of scripts must be in the .PRM extension files. Each line of the list must conform to the following structure:

- a file name, from column 1 and a maximum 8 characters long,
- un témoin d'exécution automatique des scripts, entre les signes <>. Si ce témoin est <1>, l'exécution du script précède l'affichage de l'écran. Un seul script par écran peut comporter cette particularité. Ce témoin est facultatif.
- the number of the associated icon in brackets (refer to section "Associating an icon with a script" in chapter "Advanced Functions", subchapter "ClickPad").
- a label, which starts at the first non-blank character and can stretch to the end of the line.

If the lines do not respect this format they will not be taken into account..

8.3.1.2. List of Logon Scripts

The list of the logon scripts is included in the CONNEX.PRM file. When PAW is started up, a dialog box shows the list of the available logos. The file name is the file name of the two associated script files, with the .SCI extension for the logon and the .SCF extension for the logoff.

The CONNEX.PRM file contains one line for each application, in the following way:

DSMS	operating DSMS
PAC150	PACBASE release 1.5
CICST	Test CICS
APPLI	Example application

8.3.1.3. List of Application and Screen Scripts

Let us take the example of the application called APPLI. The list of the scripts to this application is included in the APPLI.PRM file. The structure of this file is similar to that of the CONNEX.PRM file, for example:

PRTFOLD	Printing of a folder
NEWFOLD	Creation of a folder using an existing folder

The scripts linked to this application can be activated at any time between the execution of APPLI.SCI and of APPLI.SCF. The scripts which correspond to the lines of the APPLI.PRM file are PRTFOL.SCR and CREATD.SCR.

The screen scripts are referenced in a file whose name is that of the screen revamping DLL file, with a PRM extension. The scripts linked to the DO0000 screen are referenced in the DO0000.PRM file in the following way:

SEARCH <1>	Search of a technical package
INITDO	Screen initialization

8.3.2. Scripts and DDE

Scripts make it possible to set up dialogues with a DDE server via the following commands:

- Opening of a dialogue on a given topic with a server (DDEINIT),
- Request or supply of data to the server (DDEREQUEST and DDEPOKE),
- Request of end of conversation (DDETERMINATE),
- Modification of the display of the DDE server window.

8.4. Script Structure

The structure of a script is:

```
PROG            Script-name
declaration(s)
BEGIN
instruction(s)
END
```

The first line of the script must have the following structure

```
PROG            Script-name
```

This line declares the script's program line

The PROG keyword must be written in lower or uppercase letters. The program name is a series of letters and numbers starting with a letter and containing a maximum of 16 characters. This word must not be a reserved word. (See the list of reserved words below.)

No other information may be included on this line.

8.5. Reserved Words

Keyword and function names are reserved. They may not be used as names or as labels for programs and variables.

AND	FALSE	PROG
BEGIN	FOR	STRING
BOOLEAN	GOTO	TRUE
BREAK	IF	UNTIL
DO	INTEGER	WHILE
ELSE	NOT	
END	OR	

The following reserved words are pre-defined function names. They must be entered in uppercase letters:

CONCAT	ERROR	READ
CPYSCR	EXIT	READC
CURPOS	FCLOSE	RPARAM
DATAELGET	FOPEN	SCREENID
DATAELGETAT	FREAD	SCRSHOW
DATAELGETPOS	FWRITE	SEARCH
DATAELSET	GETPROFSTR	SEND
DDEEXEC	GETSGLOBAL	SETPROFSTR
DDEINIT	INPUT	SETSGLOBAL
DDEPOKE	INTSTR	STREXTRACT
DDEREQ	OUTPUT	WRITE
DDESHOW	PAUSE	WRITEC
DDETERM	PGMEXEC	WRITEM
DISPLAY		

8.6. Declarations

The declaration part of the program allows you to define the variables and labels used in the instruction. This part is located between the heading and the body of the program (if no variable is used, this part can be deleted).

- variable declaration:

```
Variable_type   Variable_name
```

- label declaration:

```
$label
```

8.6.1. Variable Types

There are three types of variables: integer, string and Boolean.

The declaration of a variable is written as follows:

```
            STRING      variable_name
or          INTEGER     variable_name
or          BOOLEAN     variable_name
```

Several variables of the same type can be declared on the same line:

```
STRING      name1 name2
```

is the same as:

```
STRING      name1

STRING      name2
```

8.6.2. Variable Names

A variable name is a series of alphanumeric characters, beginning with a letter. The maximum length is 16 characters. All variables used in the program must be declared. Two variables cannot have the same name. A variable name must not be a keyword or a function name (see the previous list for the reserved words).

The variable names must be followed by a blank character or by another non-alphanumeric character. A declaration ends with a return.

8.6.3. Variable Values

The value of a variable is indicated in the following way:

```
Variable_name=Value
```

Integers are between *(-maxint)* and *(maxint -1)*, *maxint* where maxint depends on the machine used (2^{15} = 32768 on IBM PC). Integers are manipulated by the operators described under "Instruction".

Character strings have a maximum length of 256 characters. They are entered between double quotes. They can include blanks. When one of the characters in the string is a double quote itself, it must be doubled.

When a string does not fit on one line, it is necessary to insert a "&", before the return at the end of the line. The rest of the string begins in column 1 of the following line. You can input only blanks or tabulation marks between the continuation character and the end of the line, if not the "&" is considered as a character in the string.

Examples:

```
STRING sEXEMPLE

sEXAMPLE="He says ""Hello""."
The contents of the string are:
He says "Hello".

sEXAMPLE="This string continues&
on the following line."
The contents of the string are:
This string continues on the following line.

sEXAMPLE="This string contains a &(ampersand)."
The contents of the string are:
This string contains a &(ampersand).
```

Booleans can only take two values: TRUE and FALSE. They can be combined with the logical operators (see below). Booleans are recognized in all conditional structures.

8.6.3.1. Test Syntax of a Boolean

Example: prog TEST01SCR:

The following script shows six different ways of testing a Boolean.

```
Boolean bBool
string stTrue
string stFalse

begin
  bBool = true
  stTrue = "bBool is TRUE"
  stFalse = "bBool is FALSE"

:          Syntax 1          :
           if ( BBool )
                    begin
                    ERROR ( "Syntax 1" ,"stTrue" ,1 )
                    end
           else
                    begin
                    ERROR ( "Syntax 1" ,"stFalse" ,1 )
                    end

:          Syntax 2          :
           if ( BBool == true )
                    begin
                    ERROR ( "Syntax 2" ,"stTrue" ,1 )
                    end
           else
                    begin
                    ERROR ( "Syntax 2" ,"stFalse" ,1 )
                    end

:          Syntax 3          :
           if ( NOT BBool )
                    begin
                    ERROR ( "Syntax 3" ,"stFalse" ,1 )
                    end
           else
                    begin
                    ERROR ( "Syntax 3" ,"stTrue" ,1 )
                    end

:          Syntax 4          :
           if ( NOT BBool == false )
                    begin
                    ERROR ( "Syntax 4" ,"stTrue" ,1 )
                    end
           else
                    begin
                    ERROR ( "Syntax 4" ,"stFalse" ,1 )
                    end
```

```
:       Syntax 5        :
        if ( NOT (BBool == true) )
                begin
                ERROR ( "Syntax 6" ,"stFalse" ,1 )
                end
        else
                begin
                ERROR ( "Syntax 6" ,"stTrue" ,1 )
                end
```

8.6.4. Labels

The labels used for branching (GOTO $XXX) must also be declared. A label contains a maximum of three numbers.

Label declaration:

```
$Label_name
```

No other information may be included on this line. All program labels must have distinct names.

8.6.5. Comments

Comments placed between colons ":", are not interpreted. They may be inserted anywhere, except in character strings. It is possible to write several lines of comments by placing a ":" at the beginning of the first line and a ":" at the end of the last line. Instructions inserted between comment delimiters are not taken into account:

Examples :

```
PROG            PROGRAM_NAME
:A COMMENT MAKE TAKE UP SEVERAL LINES :

/BEGIN END

PROG PROGRAM_NAME       :A PROGRAM HAS SEVERAL LINES
BEGIN                   :OF COMMENTS IN THE MARGIN
END

PROG PROGRAM_NAME       :THESE COMMENTS COVER THE KEY
BEGIN                   WORD BEGIN.  THIS IS AN ERROR
END

PROG                    PROGRAM_NAME
STRING                  STRING_NAME
BEGIN                   STRING_NAME="ARTHUR:COMMENTS:"
END
```

The value of the string is: ARTHUR:COMMENTS and not ARTHUR.

8.6.6. Blanks and Returns

Blanks are equivalent to tabulations and mark the end of a string of characters. In the remaining part of this chapter, blanks and tabulations will be underlined when they are required. A keyword that is immediately followed by an alphanumeric character is not recognized. For example, a heading cannot be written PROGname: A blank is required after keywords.

A return represents the end of an instruction. If several lines are required for an instruction; the character "&" may be used as a continuation character before the return. The "&" sign must be followed by a blank or a tabulation to be used as a continuation character, otherwise it is considered as part of the string.

Example :

```
INPUT("sign-on CICS","user code ",&UTI,6,1,)
```

is equivalent to:

```
INPUT("sign-on CICS","user code",UTI,6,1,)
```

When a return is used, it is possible to leave one or several blank lines, except in the IF ELSE control structure (see IF ELSE) in which a blank line is not allowed before ELSE.

8.7. The Body of the Program

This section is delimited by the keywords BEGIN and END. Between these two keywords is a sequence of instructions whose syntax is described in this chapter.

8.8. Instructions

The instruction that makes up the body of the program, consists of function calls, of GOTOs to other parts of the program, and of value assignments to expressions.

8.9. Assignments

Assignments involve giving a variable the value of an expression, as follows:

```
variable-name = expression
```

No other information may be entered on this line.

The expression can be either another variable, or an integer-type constant (string or Boolean), or a function call, or an expression where the operators are the expressions according to the rules in the following subchapter.

8.10. Expressions and Operators

The simplest form of instructions are as follows:

- variable_name
- integer
- literal
- Boolean

The script language provides arithmetic and logical operators that enable you to define expressions:

expression : expression$_A$ operator expression$_B$

Arithmetic operators combined with integer operators produce integers:

- expression$_A$	opposite of expression$_A$
expression$_A$ + expression$_B$	sum of the expressions
expression$_A$ - expression$_B$	difference of the expressions
expression$_A$ * expression$_B$	product of the expressions
expression$_A$ / expression$_B$	quotient of the expressions

The logical operators produce Boolean results (having the value TRUE or FALSE):

- exp$_A$ < exp$_B$ is TRUE if exp$_A$ is less than exp$_B$; exp$_A$ and exp$_B$ must be integers.
- exp$_A$ > exp$_B$ is TRUE if exp$_A$ is greater than exp$_B$; exp$_A$ and exp$_B$ must be integers.
- exp$_A$ == exp$_B$ is TRUE if exp$_A$ is equal to exp$_B$; exp$_A$ and exp$_B$ must be of the same type.
- exp$_A$ <> exp$_B$ is TRUE if exp$_A$ is different to exp$_B$; exp$_A$ and exp$_B$ must be of the same type.
- exp$_A$ and exp$_B$ is TRUE if exp$_A$ and exp$_B$ are true; exp$_A$ and exp$_B$ must be Boolean.
- exp$_A$ or exp$_B$ is TRUE if exp$_A$ or exp$_B$ is true; exp$_A$ and exp$_B$ must be Boolean.
- not exp$_A$ is TRUE if exp$_A$ is FALSE and vice versa; exp$_A$ must be Boolean.

8.10.1. Priority of Operators

The evaluation of expressions that use several operators is done according to a predefined hierarchical order. Two rules govern this order: the hierarchy of the mathematical operator and its location in the expression. The following operators are shown their order of priority (highest to lowest):

-	unary
* /	associative
- +	associative
< > <> ==	non associative
NOT	non associative
AND OR	associative

8.10.2. Processing of Associative Operators

To evaluate	A + B + C
use	intermediate = A + B
and the result is:	intermediate + C.

Two non-associative operators cannot be adjacent. The product of A and the sum of B and C must be written:

A*B+A*C

The parentheses "(" and ")" change the order of the evaluation. An expression between parentheses is equal to a temporary variable, the innermost parentheses are evaluated first:

A=(B+C-D)*(E/F)

is equivalent to:

intermediate_1=B+C
intermediate_2=intermediate_1-D
intermediate_3=E/F
A=intermediate_2*intermediate_3

The parentheses also make the expression more readable without modifying the evaluation order. Their use is recommended for expressions with operands of different types, for example:

(a>0) and (a<11)

8.11. The Unconditional Branching

The GOTO $XXX instruction shunts the sequence of the program and goes directly to the location marked with $XXX.

The blank between "GOTO" and "$" is optional and a line return is required.

Some branching is not allowed: a branching cannot be used to enter into an instruction block, but can be used to exit from one.

8.12. Control Structures

Control structures consist of groups of commands whose execution is conditioned.

The control structures are:

IF condition block ELSE block	WHILE condition block UNTIL condition	DO block	FOR condition block

A condition is written:

(expression)

where the expression has a Boolean value. The condition is true if and ONLY IF the Boolean value is TRUE.

The condition of the structure FOR is special, see below.

The keyword defining the structure and the condition must be entered on the same line. They may be separated by a blank.

A block of instructions is used to group several instructions to be executed sequentially, together and under the same condition. These instructions can be the control structures.

The instruction block is preceded by the keyword BEGIN and followed by the keyword END.

Branching can be done only from within the block, however, branching from outside a block to within a block is not permitted.

8.13. Expressions

A block of instructions is written as follows

```
BEGIN        instructions    END
```

A return is required at the end of the block. It can be inserted before the block, after BEGIN, before and after the instructions.

DO UNTIL

```
DO
             BEGIN
             instructions
             END
UNTIL        condition
```

This block is executed as long as the condition is false. The first part (DO) is executed at least once. Be careful to make sure that the condition eventually becomes true, or else you will create an infinite loop.

FOR

```
FOR (expressionA, variable_name,expressionB)
BEGIN
Instructions
END
```

The "variable_name" acts as a counter and therefore must be an integer. *ExpressionA* is evaluated once, and its value must be an integer, and its value defines the lower limit. *ExpressionB* is the upper limit. The counter takes the value of *expressionA*; and if *expressionB* is either greater than or equal, the block is executed. The counter is incremented and a new comparison is made with the upper limit.

If *expressionA* is greater than *expressionB*, the block is not executed, and the program continues sequentially after the block.

This structure allows you to execute the block a set number of times (*expressionB* - *expressionA* + 1). It uses a counter, which locates the current iteration. This counter can not be used as a variable in the block: it must not be modified within a block, it cannot be used as a counter for a nested FOR loop. If it is used somewhere else in the program, the old value will be lost at the beginning of the FOR loop.

IF ELSE

```
IF          condition
            BEGIN
            instructions
            END
ELSE
            BEGIN
            instructions
            END
```

When the interpreter detects this structure, it evaluates the condition and then executes the first block if the condition is true, the second if it is false.

The ELSE BEGIN instructions END is optional. In this case, the program immediately executes the sequence if the condition is false.

Only one return can be placed between the first END and ELSE.

WHILE

```
WHILE       condition
            BEGIN
            instructions
            END
```

The block is executed if the condition is true. If the condition is false on the first evaluation then the block is never executed. Be careful to make sure that the condition eventually becomes false, or else you will create an infinite loop.

Exit From a Loop (BREAK)

As soon as the interpreter detects the keyword BREAK, it exits from the nearest FOR, WHILE or DO UNTIL loop. This allows you to get out of an iterative structure prematurely - usually used in testing. A BREAK outside a FOR, WHILE or DO UNTIL loop will provoke an error.

The keyword BREAK must be followed by a return.

8.14. Functions

Functions can be called either as instructions or as expressions. That is, after the call, some still produce a value (return code). Others are designed to interact with the environment. They provide the technique to have the program communicate with the outside.

You will find examples of functions of the script language in the example scripts copied automatically in the SAMPLES directory at the PAW installation.

8.14.1. Function Call

A function call is entered using the following syntax:

```
function_name (lparameter_list)
```

Each function will be described later, as to whether or not a value is produced, and what the values are. The functions that produce values may be used either in an expression, or as an instruction, but the value of the return code will be lost. Functions that do not produce values can only be used as instructions.

8.14.2. Function Parameters

There are two different types of parameters: constant parameters and modifiable parameters. A constant parameter is not modified by the function. This can be an expression or a variable. A modifiable parameter is transformed by a function. These modifications are indicated in the functions' descriptions.

The parameters must be separated by a comma; They may be integer, string or Boolean. The string parameters must have double quotes. The number of parameters and their types are tied to the specifications of each individual function and must be respected.

Note: In the functions' descriptions, the word IN indicates that the parameter's value (modifiable or constant) is the same in input and in output. The word OUT indicates that the parameter's value (necessarily a modifiable parameter) has been modified in input. CR indicates the return code.

CONCAT

Concatenates several character strings

cr = CONCAT (Parm1,ParmM, ... ,ParmX)

Parm1	out string	string receiving the result of the concatenation
ParmM to ParmX	in string	string to concatenate
cr	integer	length of the result

NOTE: if the variable used as parameter 1 is not empty, its contents are also concatenated; the length of the resulting string must be less than or equal to 132 characters (maximum length of a character string).

CPYSCR

Copies the host screen into a PC file

This command does not interrupt the script execution; it is mainly in test mode or in case of error, in order to allow the user to determine the cause of the error. Note that all screens copied are stored in the file, which must be manually deleted.

cr = CPYSCR (Parm1)

Parm1	in string	complete name of the DOS file in the form: ("U:\PATH\PREFIX.EXT")
cr	integer	1

CURPOS

Reads the position of the cursor on the host screen

CURPOS (Parm1,Parm2)

Parm1	out integer	line number
Parm2	out integer	column number
cr	integer	1

DATAELGET

Gives the value of a data element whose code, type and rank are specified

cr = DATAELGET (Parm1, Parm2, Parm3, Parm4)

Parm1	in string	data element code	
Parm2	in string	data element type LABEL: fixed label PROTEC: modifiable protected field ERROR: error message INPUT: input field OPER: operation code ACTION: action code	
Parm3	in integer	data element rank (for a repetitive) (if Parm2 = 0 or 1, the first value found is taken into account)	
Parm4	out string	data element value	
cr	integer	1	command executed
		-1	screen unrecognized
		-2	data element not found

DATAELGETAT

Gives data about a data element whose position is specified

cr = DATAELGETAT (Parm1, Parm2, Parm3, Parm4, Parm5, Parm6)

Parm1	out string	data element code	
Parm2	out string	data element type LABEL: fixed label PROTEC: modifiable protected field ERROR: error message INPUT: input field OPER: operation code ACTION: action code	
Parm3	out integer	data element rank (for a repetitive) (if Parm2 = 0 or 1, the first value found is taken into account)	
Parm4	out integer	field length	
Parm5	in integer	line number	
Parm6	in integer	column number	
cr	integer	1	command executed
		-1	screen unrecognized
		-2	data element not found

DATAELGETPOS

Gives the position of a data element whose code, type and rank are specified

cr = DATAELGETPOS (Parm1, Parm2, Parm3, Parm4, Parm5)

Parm1	in string	data element code
Parm2	in string	data element type LABEL: fixed label PROTEC: modifiable protected field ERROR: error message INPUT: input field OPER: operation code ACTION: action code
Parm3	in integer	data element rank (for a repetitive) (if Parm2 = 0 or 1, the first value found is taken into account)
Parm4	out integer	line number
Parm5	out integer	column number

cr	integer	1	command executed
		-1	screen not recognized
		-2	data element not found

DATAELSET

Updates a data element field whose code and rank are specified

cr = DATAELSET (Parm1, Parm2, Parm3)

Parm1	in string	data element code
Parm2	in integer	data element rank (for a repetitive) (if Parm2 = 0 or 1, the first value found is taken into account)
Parm3	in string	update value

cr	integer	1	command executed
		-1	screen unrecognized
		-2	data element not found

NOTE: This function must be followed by a SEND ("ENT") in order to initiate the update.

DDEEXEC

Sends a request to the server

cr = DDEEXEC (Parm1, Parm2, Parm3)

Parm1	in integer	DDE dialogue identifier obtained through DDEINIT
Parm2	in string	server request. This line must conform with the syntax of the server application request language.
Parm3	in integer	maximum time limit (in seconds) given by the server for the execution of its request.

cr	integer	-1	unknown dialogue
		0	command not executed
		1	command executed

DDEINIT

Initializes a DDE dialogue

cr = DDEINIT (Parm1, Parm2, Parm3, Parm4)

Parm1	in string	name of the application which operates as a DDE server
Parm2	in string	dialogue topic: TOPIC of the DDE documentation
Parm3	in string	command line to be executed if the target application of the dialogue is not activated For WINWORD, the parm2 must be the same as the start-up parameter used in parm3. The parm3 can also be used to activate another application.
Parm4	in string	display type: SHOW displays the window with its current size and position NORMAL displays the window. If it is iconized or at its maximum size, it is displayed with its initial size HIDE hides the window ICON iconizes the window MAX displays the maximum size of the window If this parameter is given the wrong value, value SHOW is taken into account.
cr	string or integer	identifier of the DDE dialog if initialization OK, 0 otherwise

DDEPOKE

Forces the update of data in the server

cr = DDEPOKE (Parm1, Parm2, Parm3, Parm4, Parm5)

Parm1	in integer	DDE dialogue identifier obtained through DDEINIT	
Parm2	in string	object to be updated. (ex: an EXCEL cell or a WINWORD bookmark)	
Parm3	in string	update value	
Parm4	in string	Data format given (format: CF_TEXT)	
Parm5	in integer	maximum delay (in seconds) given by the client for the execution of its request.	
cr	integer	-1	unknown dialogue
		0	command not executed
		1	command executed

DDEREQ

Requests data from the server

cr = DDEREQ (Parm1, Parm2, Parm3, Parm4, Parm5)

Parm1	in integer	DDE dialogue identifier obtained through DDEINIT
Parm2	in string	object whose value is requested (ex: an EXCEL cell)
Parm3	out string	character string which contains the requested value
Parm4	in string	Data format (format: CF_TEXT)
Parm5	in integer	maximum delay (in seconds) given by the client for the execution of its request

cr	integer	-1	unknown dialogue
		0	command not executed
		1	command executed

DDESHOW

Modifies the display of the server window

cr = DDESHOW (Parm1, Parm2)

Parm1	in integer	DDE dialogue identifier obtained through DDEINIT
Parm2	in string	display type: SHOW displays the window with its current size and position NORMAL displays the window. If it is iconized or if at its maximum size, it is displayed with its initial size HIDE hides the window ICON iconizes the window MAX displays the maximum size of the window If this parameter is given a wrong value, value SHOW is taken into account.

cr	integer	-1	unknown dialogue
		0	command not executed
		1	command executed

DDETERM

Closes the DDE dialogue

cr = DDETERM (Parm1)

Parm1	in integer	DDE dialogue identifier obtained through DDEINIT. This command is **compulsory.**

cr	integer	-1	unknown dialogue
		0	command not executed
		1	command executed

DISPLAY

Displays the host screen

cr = DISPLAY()

This causes an interruption in the script execution. It is useful in testing scripts, and in error conditions, it will allow the user to see the problem screen.

No parameters

cr	integer	1

ERROR

Displays a message box

cr = ERROR (Parm1, Parm2, Parm3)

Parm1	in string	box caption
Parm2	in string	error message
Parm3	in integer	keys:

Windows (English)
1 OK
2 OK Cancel
3 Abort Retry Ignore
4 Yes No Cancel
5 Yes No
6 Retry Cancel

cr integer value of the key pressed by the user:

Windows (English)
1 OK
2 Cancel
3 Abort
4 Retry
5 Ignore
6 Yes
7 No

EXIT

Ends the script execution

EXIT (Parm1)

Parm1	in integer	value different from 0 and 1

FCLOSE

Closes a file

cr = FCLOSE (Parm1)

Parm1	in string	file name, possible with the complete path
cr	integer	index of file closing

FOPEN

Opens a file

cr = FOPEN (Parm1, Parm2)

Parm1	in string	file name, possible with the complete path
Parm2	in string	opening mode ('r' or 'R' for read, 'w' or 'W' for write and 'a' or 'A' for add)
cr	integer	index of the file opening (greater than zero) if the opening is OK. Otherwise, error code: -1: incorrect opening mode -2: too many opened files -3: other error

FREAD

Reads a file

cr = FREAD (Parm1, Parm2)

Parm1	in string	file name, possible with the complete path	
Parm2	out string	string receiving the field that has been read. The delimiter is the cr character or the end of file	
cr	integer	-2	error (see error messages)
		-1	end of file
		>= 0	number of characters read (if cr = 0, the line read is empty)

FWRITE

Writes a file

cr = FWRITE (Parm1, Parm2, Parm3)

Parm1	in string	file name, possible with the complete path
Parm2	in string	character string to be written
Parm3	in string	number of returns before the writing
cr	integer	index of file opening

GETPROFSTR

Alimentation avec un des éléments du fichier ENVIRON.PRM

cr = GETPROFSTR (Parm1, Parm2, Parm3)

Parm1	in string	section of ENVIRON.PRM file: PAW, COMM, PATHS, CUSTOM, SYSTEM
Parm2	in string	element of ENVIRON.PRM file. This element must be preceded by the section to which it belongs. The number in parentheses is the line number in the ENVIRON.PRM file. For the **PAW** section: Version (001), UserLang: user workstation language (001), DevLang: developer workstation language (001), Target System: target system (007), LearningMode: learning mode (023), WorkStation: type of workstation (024). For the **COMM** section: ProgramName: name of communication manager (002), ParamName: associated parameters file (002). For the **PATHS** section: Exe: path of .exe files (003), User: user path (004), ListDlls: value lists paths (005), Developer: developer's path (006), Communication: GSCOM files directory (027) Parameters: parameter files path (028), Scripts: scripts path (030), TText: DSMS text editor path (034). For the **CUSTOM** section: IconDll: icons file (020). For the **SYSTEM** section: PollingRate: interrogation delay (020), ScreenAdjust: automatic adjustment of the screens (021), ExternalNames: external names (022), ScriptWindow: display the script window in test mode (031).
Parm3	out string	string which receives the specific element
cr	-1 0	failure success

GETSGLOBAL

Consults a global variable.

cr = GETSGLOBAL (Parm1, Parm2)

Parm1	in string	name of global variable to consult.
Parm2	in string	value of this variable.
cr	1	command executed (variable found)

INPUT

Creates a dialogue box

cr = INPUT (Parm1,ParmN,ParmN+1,ParmN+2,1ParmN+3,&
ParmN,ParmN+1,ParmN+2,1ParmN+3,&
...
ParmN,ParmN+1,ParmN+2,1ParmN+3)

This function has a variable number of parameters:

Parm1	in string	name of the dialogue box

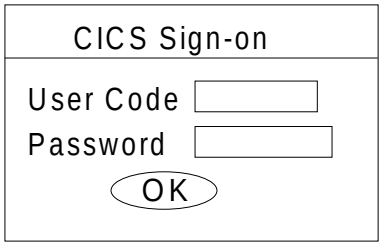
The parameters are grouped in fours, each group corresponding to a dialogue line in the window; the number of lines is not limited.

ParmN	in string	label of the line
ParmN+1	out string	variable receiving the user's answer
ParmN+2	in integer	length of the answer
ParmN+3	in integer	display user input on the screen: 1 YES 2 NO (e.g. passwords)
cr	integer	1 OK

Example :

INPUT("Sign-on CICS","User Code",UTI,6,1,"Password",PAS,8,2)

This command builds the following dialogue box:



This box includes two input fields: the user code (UTI), six characters which are displayed, and the user password (PAS), eight which are not displayed. The string for "CSIS Sign-on" is truncated after 50 characters.

INTSTR

Converts an integer into a string

cr = INTSTR (Parm1, Parm2)

Parm1	out string	resulting string
Parm2	in integer	integer to convert
cr	integer	length of the result

OUTPUT

Displays a message on the PC screen

cr = OUTPUT (Parm1, Parm2)

Parm1	in string	message to display
Parm2	in integer	display type
0 or >6		the message is displayed in the connection box in order to identify the current connection phase.
1 à 6		message box overrides the connection box; see the documentation on the ERROR function for a complete description of these values.
cr	integer	1 if parameter 2 = 0; see the ERROR return codes in the other cases.

PAUSE

Sets a time period between two commands

cr = PAUSE (Parm1)

Parm1	in integer	time in seconds
cr	integer	1

PGMEXEC

Starts up a WINDOWS application

cr = PGMEXEC (Parm1, Parm2)

Parm1	in string	command line containing the name of the program to run and possibly one or several parameters	
Parm2	in string	display type: SHOW: displays the window with its current size and position NORMAL displays the window. If it is iconized or if at its maximum size, it is displayed with its initial size HIDE hides the window ICON iconizes the window MAX displays the maximum size of the window. If this parameter is given a wrong value, value SHOW is taken into account.	
cr	integer	0 1	command not executed command executed

READ

Reads a character string

cr = READ (Parm1, Parm2, Parm3, Parm4)

Parm1	in integer	line
Parm2	in integer	column
Parm3	out string	string receiving the result
Parm4	in integer	length of the field to read

cr	integer	1	read OK

READC

Reads a field on the screen at the cursor position

cr = READC (Parm1,Parm2)

Parm1	out string	string receiving the result
Parm2	in integer	length of the field to read

cr	integer	1	read OK

RPARAM

Reads PACBASE identification connection parameters during the script execution

cr = RPARAM (Parm1, Parm2)

Parm1	in integer	parameter type
Parm2	out string	string receiving the parameter

cr	integer	1	OK
		2	invalid value for Parm1

Parameter type

0	user code
1	password
2	database code
3	library code
4	session number (9999: current session)
5	session type (H: frozen session/T: test session)
6	DSMS product code
7	DSMS Change number

SCREENID

Supplies a character string with the code of the screen DLL

SCREENID (Parm1)

Parm1	out string	string which receives the DLL code

SCRSHOW

Displays a PAW screen which was previously hidden

cr = SCRSHOW (Parm1)

Parm1	in string	display type: SHOW displays the window with its current size and position NORMAL displays the window. If it is iconized or if at its maximum size, it is displayed with its initial size HIDE hides the window ICON iconizes the window MAX displays the maximum size of the window If this parameter is given a wrong value, value SHOW is taken into account	
cr	integer	0	command not executed
		1	command executed

SEARCH

Searches a character string on the screen

cr = SEARCH (Parm1,Parm2,Parm3,Parm4,Parm5)

Parm1	out integer	line where the search begins (0: the search is performed on all lines)	
Parm2	out integer	column where the search begins (0: the search is performed on all columns)	
Parm3	in string	string to search	
Parm4	in integer	length of the string to search	
Parm5	in integer	maximum duration of the search in seconds	
cr	integer	0	string not found
		1	string found

NOTE:

Parm4 is not systematically used:

- if the length parameter is zero or is greater than the length of the string to search , it is ignored;
- if the length parameter is less than the length of the string to search, the function searches the sub-string with a length equal to the length parameter;
- the value 0 is ignored.

EXAMPLE:

SEARCH (ln, cl , "AAAA" , 12 , 10) searches "AAAA"
SEARCH (ln, cl , "AAAAAAAA" , 4, 10) searches "AAAA".

SEND

Sends the value of a key

cr = SEND (Parm1)

Parm1		in string	key to send
cr	integer	1	normal send
		2	unknown key
		3	no answer from host

KEYS FOR ALL HARDWARE

ENT	enter
GCHE	cursor to the left
DRTE	cursor to the right
HAUT	cursor upwards
BAS	cursor downwards
TAR	tabulation backwards (on formatted screen)
TAV	tabulation (on formatted screen)
PF01 à PF24	Function keys

WITH A 3270 PROTOCOL:

ATTN	attn
ATTR	attributes
CLS	clear screen
CURS	select cursor
DEL	delete one character
DUP	dup
EEOF	erase EOF
ERA	erase
HOME	return to beginning of line
PA1	PA1
PA2	PA2
PA3	PA3
PRN	print
REST	restore
RSET	reset
SYST	system call

WITH A VIP-QUESTAR PROTOCOL:

CLS	local screen clear + transmission
FLC	local screen clear + transmission
HOME	first character on the screen

SETPROFSTR

Modification of the work environment (Parm1 and Parm2) with the contents of Parm3.

Note: the ENVIRON.PRM file is not modified. Only the values in memory are affected.

cr = SETPROFSTR (Parm1, Parm2, Parm3)

Parm1	out string	section of ENVIRON.PRM file: PAW or PATHS.
Parm2	out string	element of ENVIRON.PRM file. This element must come before the section which it belongs to. The number in parentheses is the line number in the ENVIRON.PRM file. For the **PAW** section: UserLang: user language (001). For the **PATHS** section: ListDlls: values lists path (005), Scripts: scripts path (030).
Parm3	in string	string containing a value updating an element of the ENVIRON.PRM file.
cr	-1 0	failure success

SETSGLOBAL

Memorization of a global variable

cr = SETSGLOBAL (Parm1, Parm2)

Parm1	out string	name of the variable to be positioned or modified (if it already exists).
Parm2	in string	value to be assigned to this variable.
cr	1	command executed

STREXTRACT

Extraction of a string segment

cr = STREXTRACT (sOrigin, iBeginning, iLength, sTarget)

cr	integer	length of the extracted string

WRITE

Writes a character string on the screen

cr = WRITE (Parm1, Parm2, Parm3 , Parm4)

Parm1	in integer	line
Parm2	in integer	column
Parm3	in string	character string to write
Parm4	in integer	MDT bit modification (3270 mode) display the string in test mode
0	no modification of the MDT, string displayed	
1	modification of the MDT, string displayed	
2	no modification of the MDT, string not displayed	
3	modification of the MDT, string not displayed	

cr	integer	1	write OK
		2	error on write

NOTE: This function must be followed by a SEND ("ENT") in order to initiate the update.

WRITEC

Writes a character string on the screen at the cursor position

cr = WRITEC (Parm1, Parm2)

Parm1	in string	string to write
Parm2	in integer	MDT bit modification (3270 mode) display the string in test mode
0	no modification of the MDT, string displayed	
1	modification of the MDT, string displayed	
2	no modification of the MDT, string not displayed	
3	modification of the MDT, string not displayed	

cr	integer	1	write OK
		2	error on write

(Note that the write may not be executed when the cursor is not on an input field)

NOTE: This function must be followed by a SEND ("ENT") in order to initiate the update.

WRITEM

Writes a character string in non-formatted mode - for Bull hardware

cr = WRITEM (Parm1,Parm2)

Parm1	in integer	character string to write	
Parm2	in integer	in test mode:	0 = string displayed 2 = string hidden

cr	integer	1	write OK
		2	error on write

NOTE: This function must be followed by a SEND ("ENT") in order to initiate the update.

8.15. Errors

When the interpreter detects an error, it terminates the script's execution and sends a message which indicates the number of the erroneous line and if possible a diagnostic message.

Errors may be divided into three groups: source code errors, syntax errors and execution errors. The first two groups involve errors found in reading the source code: variable types and their utilization, branches and their labels, and syntax errors. The second category involves errors found in the source of the execution of the script.

8.15.1. Source Code Errors

The principal characteristic of source code is that the interpreter will send an error message to the screen should an error be detected and then continue. The script however is not executed.

The interpreter checks for compatibility between variable and constant types, the validity of expressions and of parameters. Constraints imposed upon operator types, parameters or predefined functions, counters or FOR loops and conditions are documented earlier in this chapter.

Several tests are performed on variables: using a variable which was not initialized directly or by assignment; using a variable that is used as a parameter in a predefined function causes an error, as well as using an undeclared variable. When variables are declared and not used, a warning message is issued though this causes no error.

Unconditional branching is strictly controlled: labels must be defined; branching inside an instruction block causes an error. When declared labels are not used, a warning message is issued, though this causes no error.

8.15.2. Syntax Errors

When a syntax error is encountered, the interpreter issues an error message and returns control to the user. The error message indicates the number of the line where the error was detected. Note that the line number refers to the line on which the inconsistency problem was detected, and not necessarily where the correction needs to be made.

EXAMPLE:

```
1       PROG PROGRAM_NAME
2       STRING SELECTOR
3       BEGIN
4               SELECTOR="OPEN"
5               IF (SELECTOR=="OPEN")
6                   BEGIN
7                       WRITEM ("RUN")
8                   END
9               ELSE
10              BEGIN
11                  WRITEM ("WAIT")
12                  PAUSE (25)
13      END
```

In this example, the error will be detected on line 13, whereas the error is due to a missing END at line 12: the interpreter cannot know the number of lines contained in the ELSE block before it reads line 13.

Syntax errors are often due to:

- missing or superfluous block delimiters (BEGIN or END);
- operators which should be on a new line but are not;
- line skip before an ELSE;
- unmatched parentheses;
- missing parenthesis in conditions;
- final END missing.

8.15.3. Errors during Execution

Though many errors may be detected before an actual execution, there are problems which may only appear when the script is run.

These problems depend on the length and intricacies of the script, as well as on the correctness of algorithms. Problems due to limitations of the PC's memory and the maximum time allowed for routine execution depend on the hardware and operating system used: for instance, an endless loop could cause an error or may even impact on the PC's performance.

The arithmetic calculations may go over the absolute maximum value for an integer (32767). This can lead to erroneous results, but not always in an obvious way (the calculation of de a+b-c, with a, b and c large can cause an error in the calculation of a+b, while the calculation of a+(b-c) can encounter no problems). The interpreter stops the execution in the event of such a problem (the error message does not give the number of the line to blame).