IBM VisualLift for MVS, VSE, VM, & OS/390 User's Guide

SC33-6691-02

Version 1 Release 1

IBM VisualLift for MVS, VSE, VM, & OS/390 User's Guide

SC33-6691-02

Version 1 Release 1

Note!

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

Third Edition, June 1996

This edition applies to IBM VisualLift for MVS, VSE, VM, & OS/390 (5648-109). and to all subsequent releases and modifications until otherwise indicated in new editions.

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

A form for readers' comments is provided at the back of this publication. If the form has been removed, address your comments to:

IBM Corporation
Attn: Dept. ECJ - BP/003D
6300 Diagonal Highway
Boulder, CO 80301,
U.S.A.

or to:

IBM Deutschland Entwicklung GmbH
Department 3248
Schoenaicher Strasse 220
D-71032 Boeblingen
Federal Republic of Germany

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

Contents

Figures

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 IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

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

Trademarks

The following terms, used in this publication, are trademarks of the IBM Corporation in the United States or other countries or both:

Common User Access
CUA
IBM
IBM DOS
MVS/ESA
OS/2
OS/390
PS/2
Systems Application Architecture
SAA
S/370
S/390
VGA
VM/ESA
VSE/ESA
VisualLift
XGA

The following terms, used in this publication, are trademarks or registered trademarks of other companies:

Microsoft, Windows, Windows/NT and the Windows 95 logo are trademarks of Microsoft Corporation

MS-DOS	Microsoft Corporation
EXTRA!	Attachmate Corporation
RUMBA	Wall Data Incorporated
DCA	Digital Communication Associates, Inc.
IRMA	Digital Communication Associates, Inc.
NetSoft	NetSoft
DynaComm/Elite	Future Soft Engineering, Inc. and NetSoft

Chapter 1. Introduction

Welcome to IBM VisualLift for MVS, VSE, VM & OS/390 *the* user interface modernizer for S/370 and S/390 applications. With VisualLift you can design an OS/2 or Windows workstation user interface for MVS, VSE, VM, and OS/390 applications.

To work with VisualLift is easy and fast: for novice users the graphic editor is suitable to design the workstation windows, and for expert users the textual representation is the productive way to define workstation windows. Before you start, please read the following information carefully - you will benefit later when using VisualLift.

Information

The information provided with VisualLift is structured according to the various stages of working with VisualLift. The following information is provided with VisualLift:

Figure 1. The Information Delivered with VisualLift

Type	Purpose
User's Guide	Use the VisualLift User's Guide to get a basic understanding what VisualLift is and how to use it.
Help	Use help information when using VisualLift. How to perform a task or step are the questions which are answered by the VisualLift help information. Help is available anywhere within VisualLift.
Online Solution Guide	Use the VisualLift Online Solution Guide to get information about definitions and examples of controls, and about mapping and mapping modes. It is the main source for designing PWS windows. See page 28 about how to access the Solution Guide.
Reference	Use the VisualLift Reference when • Working in textual mode • Writing application provided routines (routines for your own applications) • Working with event files. The VisualLift Reference provides answers for expert users. The VisualLift Reference is located in the VisualLift Folder.
Samples	Use the VisualLift Samples as a base how to learn writing window definitions in textual mode. The VisualLift Samples are located in the SAMPLES project.

VisualLift User's Guide

The VisualLift User's Guide provides the information necessary to use VisualLift. The information is conceptual and explanatory. For example, concepts of VisualLift are described as well as a step by step sample on how to Visual*Lift* a host application. The following provides an overview of the various chapters:

1 **Introduction** explains how the information of VisualLift is organized (who uses what with which skill).

2 **Chapter 2, "What is VisualLift?"** explains in form of an overview what VisualLift is all about.

3 **Chapter 3, "VisualLift Concepts"** explains the fundamentals of VisualLift.

4 **Chapter 4, "Installation"** contains and refers to the information needed to install the VisualLift OS/2 ADE and RTE as well as the VisualLift Windows RTE.

5 **Chapter 5, "Getting Started"** explains the purpose of the objects residing in the VisualLift folder.

6 **Chapter 6, "Using VisualLift"** is the main chapter of this book. Here you find all information how to work with VisualLift. A scenario is used to explain how to Visual*Lift* an application.

7 **Chapter 7, "VisualLift RTE for Windows"** informs you about the deviations of the Windows RTE version of VisualLift compared to the OS/2 RTE version of VisualLift.

8 **Chapter 8, "VisualLift and Automated Build Process"** describes commands to perform an automated build process for the PWS windows and the application itself.

9 **Chapter 9, "VisualLift Hints and Tips"** is the chapter where the tricks of the trade are unfolded.

10 **Chapter 10, "Problem Determination"** explains how to proceed in case of an error.

Help Information

VisualLift comes with extensive help. The VisualLift help information is task oriented and always available. You can obtain help at any time by:

- Pressing **F1** anywhere in VisualLift
- Selecting the menu bar **Help** and the respective pull-down
- Pressing the **Help** push button.

VisualLift Online Solution Guide

The Solution Guide offers information and help for the novice **and** experienced user.
It provides

- Examples for the design of workstation windows.
- The definition of controls with the graphic editor.
- The textual representation of controls.

- Information on the various mapping modes.
- Parts of host situations and suggested workstation solutions.
- Examples of different types of events.

VisualLift Reference

The VisualLift Reference is designed for the experienced user and provides detailed technical information on how to:

- Define controls via tags and parameters (textual mode)
- Write application provided routines
- Define events.

You can use the VisualLift Reference in many ways:

- Double-click on the VisualLift Reference icon to view the contents
- Type FCLREF in the OS/2 command line to view the contents
- Type FCLREF *parameter* in the OS/2 command line to view the information for a parameter, for example: FCLREF LAYOUT

VisualLift Samples

Samples for each VisualLift control are located in the SAMPLES project within the VisualLift Workbench. Samples for application provided routines are located in *fclroot*[1] \FCL\USEREXIT\SAMPLE.C

Skills for VisualLifting

The *ideal* person to Visual*Lift* a host application has knowledge of the tasks listed in Figure 2. The left column of the table lists the desirable skills. The right column lists alternatives if these skills are not available. Alternatively several persons with these skills may work in a team to exploit the benefits of VisualLift.

Figure 2. The Tasks of a VisualLifter

Task	**Where to Start? What to do?**
Using OS/2 or Windows	The *OS/2 Tutorial* is recommended to learn the basic interaction techniques. For a successful start using Windows study the tutorial delivered with Windows. If you have no Windows experience, consider using OS/2.
User interface design	*Object-Oriented Interface Design, SC34-4399*, provides guidelines to adhere to SAA Common User Access. See also the VisualLift Solution Guide.
Host application knowledge	The person who Visual*Lifts* the host application should have basic knowledge of the host application. The contribution of a subject matter expert of the host application (during the design phase) will improve the quality of the Visual*Lifted* application.
Programming	To write application provided routines is optional. If you want to use the sophisticated functions of VisualLift within such routines the C or C++ programming language is mandatory. The programming interfaces of VisualLift are described in the VisualLift Reference.

Who Should Read This Book?

The person who wants to Visual*Lift* a host application should read all chapters and all tasks described in this book.

The person who uses the Visual*Lifted*[2] application should read Chapter 4, “Installation” on page 23 - especially the installation of the RTE - and the tasks concerning *Run VisualLifted Application* (described in “VisualLift Tasks” starting on page 31).

OS/2 and Windows Terminology

All information for VisualLift is written in terms following the *Object-Oriented Interface Design, SC34-4399-00*. OS/2 follows these guidelines - the Windows user interface does not comply to this standard. Refer to the *glossary* “Glossary” on page 105.

[1] The notation *fclroot*\FCL\USEREXIT\SAMPLE.C indicates the drive, directory(ies), and file.

The environment variable *fclroot* is the placeholder for the drive and directory where VisualLift was installed, for example *D:**LIFT*. For detailed information on environment variables refer to “Environment Variables” on page 101.

[2] The term Visual*Lifted* marks a host application as modernized with a new, Visual*Lifted* user interface.

Chapter 2. What is VisualLift?

VisualLift is a tool to modernize the user interface of existing host applications. The new user interface is located on the workstation. The host application remains untouched.

The difference is that the user interface interacts with the proven host application via familiar workstation controls, such as push buttons, check boxes, notebooks, and many more. Figure 3 and Figure 4 show the user interface of a host application and the corresponding user interface of the Visual*Lifted* window.

Screen

```
------------------------------ IC Print Menu ----------------------- Menu 1 of 2

File to be printed ===> $T$E$M$P  EOSPRINT  A    Print as ===> EOSPRINT

Print information:
   Printer          ===> BOE16044   ( Press PF1 for printer selection )
   Classification   ===> UNC        ( UNC, IUO, IC, or ICR )
                    ===> S          ( A = all pages, S = Separator page only )
   Document Type    ===> NOCC       ( NOCC,FLAT,CC,TRC,DCF,GML,Bookie, RFT )
   Language         ===> US         ( Press PF1 for language selection list )
   Distribution     ===> 71032-06   ( Distribution info on separator page )
   Copies           ===> 1          ( 1 - 99 )
   Lines per inch   ===> 8          ( 1 - 18, (6, 8, 12 for DCF) )
   Fonts (max 4)    ===> X0GT2A                          ( PF1 for list )
   Top margin       ===> YES        ( 0 to 9.99 inch, Yes, or No )
   Left margin      ===> YES        ( 0 to 9.99 inch, Yes, or No )
   Duplex           ===> YES        ( No, Yes, or Tumble )
   Orientation      ===> 0          ( 0 or N, 90 or E, 180 or S, 270 or W )
   Page mode        ===> NO         ( No = line mode, Yes = all points addr. )
   Print/Save/Disp  ===> P          ( P = Print, S = Save Output, D = Display )

Command ===>

Enter=Run PF1=Help 3=Exit 4=Check         8=Next 9=Def 10=Save 12=Run&Exit
```

Figure 3. The Host

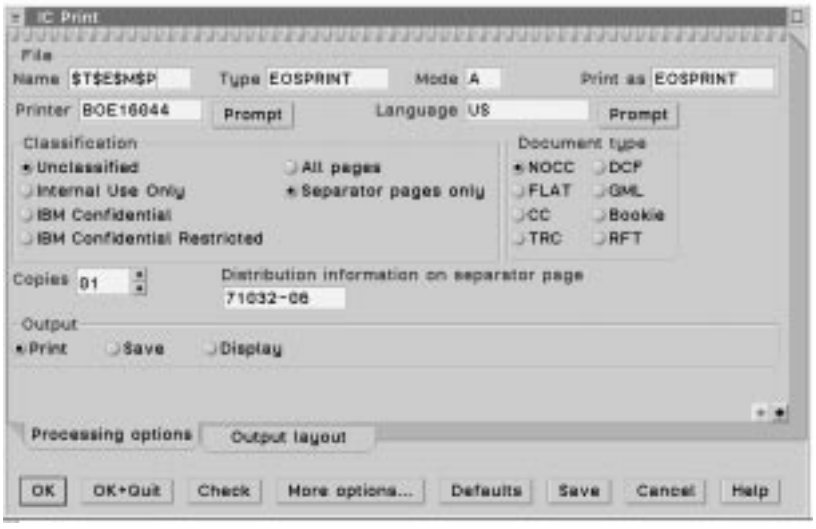


Figure 4. The VisualLifted Window

VisualLift and Host Applications

A Visual*Lifted* application consists of the *unchanged* host application and the user interface at the workstation. All logic and data of the host application remains on the host. VisualLift adapts to the host application in such a way that the host application thinks it is talking to 3270 terminals. The user of the host application has now the choice to work with:

- The traditional host user interface
- The OS/2 user interface
- The Windows user interface.

3270 Terminals and Workstations

No matter whether you are using 3270 terminals or VisualLift equipped workstations: you can use the same host application concurrently on a 3270 terminal or on a workstation. The coexistence of 3270 terminals and workstations guarantees you to exchange 3270 terminals by workstations step by step.

VisualLift is designed to work with S/370 and S/390 hardware. VisualLift supports heterogeneous environments where workstations, either stand alone or connected via a LAN, are connected to a S/370 or S/390 host running MVS, VSE, VM, or OS/390.

Candidates for VisualLift

All host applications based on the 3270 user interface technology are candidates for VisualLift - even those host applications where the source code is no longer available can be Visual*Lifted*.

Regardless whether the host application runs under MVS, VSE, VM, or OS/390 - you can modernize their user interface with VisualLift. If you are running, for example, VSE and VM based applications, you can give both, VSE applications *and* VM applications, one *consistent* user interface with VisualLift.

VisualLift Characteristics

- Increases the end-user productivity by using workstation controls
- Maintains the same reliability, availability and serviceability characteristics as the underlying host applications
- Provides consistency of the workstation user interface across host applications and workstation applications
- Requires no changes to the host application
- Uses an object oriented toolkit for developing the workstation user interface without programming
- Supports national languages on the workstation interface without changing the host application
- Organizes the user interfaces of host applications in folders on the desktop.

Chapter 3. VisualLift Concepts

Hint

This chapter describes the conceptual and technological background of VisualLift and may be skipped for first reading.

VisualLift is a user interface modernizer for host applications. Visual*Lifting* a host application is performed in the application development environment (ADE). The ADE is available on OS/2. To run a Visual*Lifted* application, the run-time environment (RTE) is used. The RTE is available either for OS/2 or Windows. Therefore Visual*Lifting* a host application has to be done in the ADE - independent of whether it will run in the OS/2 or Windows environment. Figure 5 illustrates the principle.

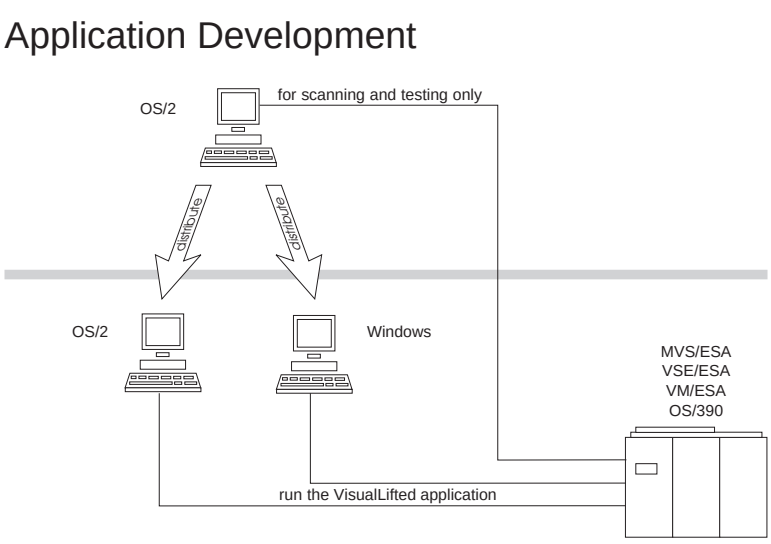


Figure 5. The Components of VisualLift

The advantage of this concept? A host application is Visual*Lifted* on an OS/2 equipped workstation. Once the host application is Visual*Lifted* it can be distributed either to OS/2 or Windows based workstations. Your users have the choice to select the preferred user interface for their host application.

Application Development Environment

The ADE is the part of VisualLift where the new user interface for the host application is designed, created, and bundled. The ADE becomes visible to you in the form of the VisualLift Workbench. The VisualLift Workbench is the place from which Visual*Lifting* starts and all VisualLift tasks to Visual*Lift* a host application are performed.

Tasks

The tasks to Visual*Lift* a host application are shown in Figure 6. Perform all tasks in the shown sequence and the Visual*Lifted* application will be successfully created. Chapter 6, “Using VisualLift” on page 31 provides detailed information on how to perform the tasks.

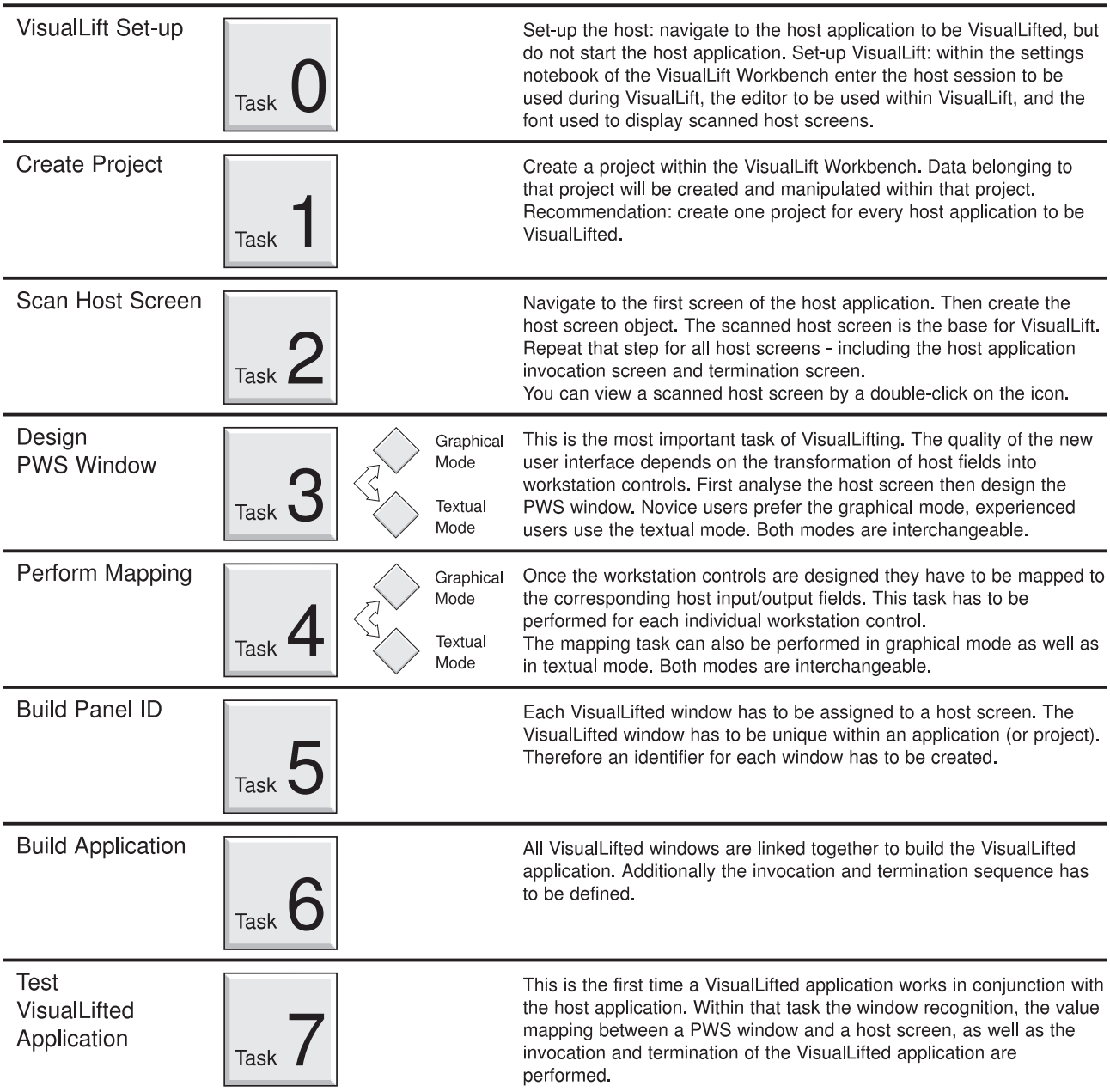


Figure 6. Tasks to VisualLift a Host Application

Graphical and Textual Mode

As shown in Figure 6 on page 8, the steps *Design PWS Window* and *Perform Mapping* can be performed either in graphical or in textual mode. Basically the graphical mode is for the novice user - it is more intuitive, whereas the textual mode is for the experienced user - it is more productive.


smpstat

is the graphical representation of the window **smpstat**. A double-click on the icon opens the VisualLift graphic editor where controls can be created or changed via dialogs and drag and drop operations.


smpstat

is the textual representation of the window **smpstat**. A double-click on the icon opens the text editor where controls can be created or changed by editing and also by cutting and pasting text elements.

Both modes are interchangeable. You can transform the graphical mode into the textual mode - or vice versa. This option provides an easy transition from the graphical representation to the declarative language - the textual representation. The structure of the language is described in the VisualLift Reference. The VisualLift Reference and the delivered samples are the base to write window definitions.

VisualLift does **not** synchronize both formats automatically:

1. The text editor runs in a separate OS/2 session and cannot be controlled by VisualLift
2. It is sometimes desirable to maintain both formats separately. For example, you can use the graphic editor to investigate a design alternative, and keep the original textual format as backup.

   When using both modes be very careful when switching back and forth. You must avoid overwriting changes you applied in one mode by changes you applied in the other mode.

Be aware

To avoid accidental overwriting of changes performed in the other mode, always use the **File** rather than the **Save** action to store changes in either format.

Arrangement of Window Contents

Unlike most other user interface editors, the VisualLift graphic editor does not allow to define the sizes and positions of controls in a window in terms of pixels, dialog units, or (x,y) coordinates. With the VisualLift graphic editor the controls are arranged using **relative layouting** rather than specifying absolute sizes and positions.

The advantage of a relative layout is greater flexibility when modifying the window. Adding, removing, or changing the location or size of controls within a window usually makes it necessary to reconsider the layout of the other controls. After that kind of modifications VisualLift uses relative layout information to automatically re-arrange all controls of the window. This also applies to other modifications that may affect the layout at run-time, for example, using different fonts or text strings with a variable length. Different fonts result in different control sizes, especially for the different target systems OS/2 and Windows. If text strings are used where the

content depends on an actual value of the host application or changes due to multiple national languages, those different values also influence the window layout. Also in those cases VisualLift will adapt the window layout to the actual requirements, using the relative layout information that is defined for the window. This adaptation is performed dynamically at application run-time.

You define relative layout using the VisualLift layout controls and attributes. **Layout controls** define the location of controls within the window in terms of lines and columns. The layout controls

- *ColumnHeadingGroup*
- *ColumnHeading*
- *ColumnGroup*
- *Column*
- *Line*
- *Group*
- *Space* and
- *Scale*

are available in the templates folder of the graphic editor. For example, dragging a *Scale* box into your window and then a set of *EntryFields* into the *Scale* will reduce or expand the spaces between the *EntryFields*, depending to the X- and Y-factors that you specify for *Scale*.

Layout attributes define how controls are aligned and expanded horizontally and/or vertically. The layout attributes

- *Justify* and
- *Expand*

can be specified in the *Attributes* notebook that is displayed when you double-click on a control in the graphic editor. For example, specifying *Expand=Horizontal* for an *EntryField* will expand the horizontal size of the *EntryField* to the available size, even if the specified width is smaller.

Run-Time Environment

The RTE is the *executing* part of VisualLift. It is used to run the Visual*Lifted* application. The RTE is the bridge between the user interface on the workstation and the application on the host. Basically, the RTE invokes and terminates the host application, displays the workstation user interface, and manages user input.

Tasks

The tasks to run a Visual*Lifted* application are shown in Figure 7 on page 11. Chapter 6, “Using VisualLift” on page 31 provides information on how to perform these tasks.

Distribute VisualLifted Application	Task 8	Distribute the VisualLifted application and copy it to a directory where it is accessed by the VisualLift RTE. The distribution of the VisualLifted application can be done via diskettes, via LAN, or via the host.
Run VisualLifted Application	Task 9	Associate the VisualLift RTE object with the VisualLifted application. Start the VisualLifted application by a double-click on the application icon.

Figure 7. Tasks to Run a VisualLifted Application

Technological Concept

VisualLift converts the host user interface (based on 3270 technology) to a workstation user interface. The 3270 data stream is intercepted by the RTE. The RTE exchanges the screen information of the 3270 data stream with the workstation windows. Additionally, the RTE manages the user input from the workstation. The result is the new user interface on the workstation. Figure 8 illustrates the principle.

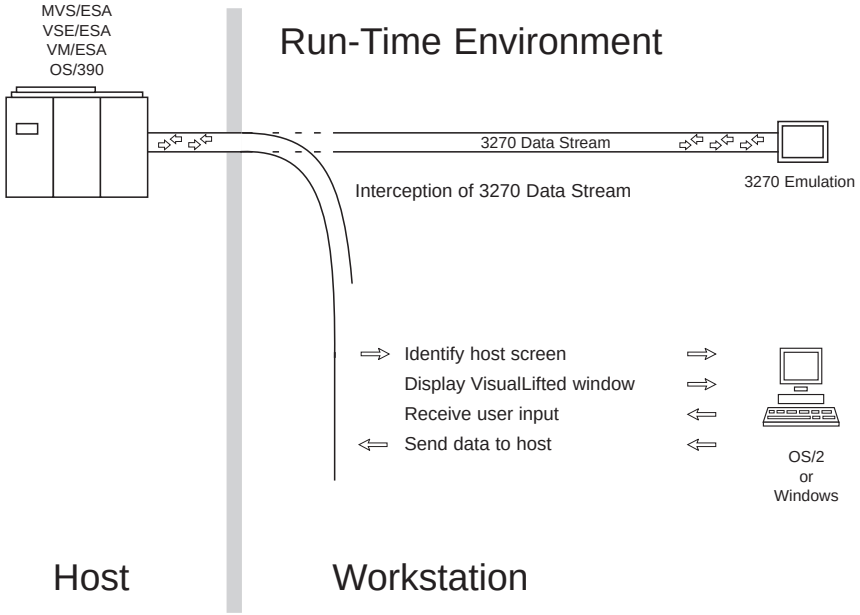


Figure 8. VisualLift and the 3270 Data Stream

Integrity

The VisualLift integrity feature guarantees that the underlying host application continues to function in *all* cases. That means, no matter where an error occurs or whatever the reason is - users are able to continue their work.

Take the case that the host application changes a screen with a new release. VisualLift recognizes this. VisualLift automatically presents the changed host screen in the 3270 emulator window. The application continues uninterrupted.

Then VisualLift switches back into normal operation with the next Visual*Lifted* window it encounters.

Application Invocation

When a VisualLifted application is started, the RTE processes the parameters that have been defined in the settings for that application.

1. Init condition parameters
   The definition of initiation conditions is optional. You can define one or more initiation conditions where each condition is characterized by a string occurring at a certain position in the host screen. Defining initiation conditions assures that the correct host screen is visible in the emulator window if the application is to be invoked. Only if all initiation conditions fall true, the application will be invoked, otherwise the invocation process is terminated with an error message
2. Invocation parameters
   The invocation *command* is placed at the specified *position* in the host screen and the *function key* is sent. After the resulting first host screen update the RTE starts its processing sequence as described in the next section. If the *system* is "CMS", the CLEAR key is sent before the specified function key. The system parameter determines also which events (see “Defining Events” on page 67) will be processed by comparing it with the system parameter that is specified in the event definition.

   The command and position may be omitted, for example, if the application is invoked by just sending a PF key. If the function key is also omitted, the run-time processing sequence starts immediately.
3. End condition parameters
   The definition of end conditions is optional. You can define one or more end conditions where each condition is characterized by a string occurring at a certain position in the host screen. The end conditions characterize the host screen allowing the RTE to recognize that the host application has ended. As a result of that (only if all end conditions fall true), the VisualLifted application also terminates.

   If no end condition is specified, the RTE keeps the application open forever, staying in its processing sequence loop, as described in the following section.

Run-time Processing Sequence

As soon as a host screen update is detected, the RTE tries to identify the host screen according to the following rules:

1. The updated host screen is **scanned** and its panel ID is generated.
2. It is checked whether there is a PWS window object having a panel ID that matches the ID of the currently active host screen. This step is called **panel identification**. The panel identification process observes the field structure of the host screen (positions and lengths of all fields, but not the field contents), the field protection states (protected/unprotected), and the extended panel ID information. For details on the panel identification process refer to “Panel Identification Process” on page 14.

   If the panel identification process finds a PWS window in the VisualLifted application with a matching panel ID, this PWS window is displayed. The user interaction with the window results in another host screen update and the

run-time processing sequence starts with step 1. If the panel identification process does not find a matching panel ID, it continues with step 3.

Note: It may happen that the currently active host screen is not updated by a user interaction, that means by an attention key (such as Enter, a PFx key, PAx key, or Clear), but asynchronously by the host application. In those cases it depends on the *DynamicRefresh* setting of the PWS window object whether the data in the PWS window are updated with the new host data or not. Only the controls in the PWS window which have the "dynamic refresh" flag set are updated with the data of the host screen.

3. It is checked whether the **application end conditions** are fulfilled (refer also to “Application Invocation” on page 12). If the application end conditions are fulfilled, the panel identification process terminates the lifted application, otherwise it continues with step 4.

4. It is checked whether the host screen should be handled as an **event** (see “Defining Events” on page 67). The conditions specified in the application event file are checked first, followed by the conditions in the system event file. The event conditions are checked bottom-to-top, that is, the last event definition in the event description file is checked first. Note that application switch events are not processed in this step, but in step 5.

   If one of the conditions is true, the RTE performs the processing that is defined for this event in the event description file (for example, an event defined with the parameters *Action=Ignore* and *Key=Enter* causes an enter key to be sent to the host). The event processing results in another host screen update and the run-time processing sequence starts with step 1. If no event could be detected, it continues with step 5.

5. It is checked whether the host screen should be handled as an **application switch event**. Although the application switch conditions are specified in the event description file, they are checked only after all other event conditions are checked, and none of them matches. Refer to “Application Switching” on page 15 for more information.

   If one of the conditions is true, the RTE closes the currently active VisualLifted application, starts the new application, and the run-time processing sequence starts with step 1. If no application switch could be detected, it continues with step 6.

6. The 3270 emulator window is restored and put to the foreground. Additionally, the title text of the 3270 emulator window changes to
   *VisualLift - **X**: Screen not recognized* (***MYAPPL***)
   where

   X is the short identifier of the 3270 session
   MYAPPL is the name of the VisualLift application

   The RTE remains in this state until the next host screen update occurs. Then the run-time processing sequence starts with step 1.

Any condition that is detected in the run-time processing sequence may be overwritten or modified within a panel ID exit routine (see “Supplying Application Provided Routines” on page 67).

Panel Identification Process

A VisualLifted application consists of one or more PWS windows, each representing a screen of the corresponding host application. Whenever a screen appears on the host at run-time, VisualLift substitutes this screen by a graphic user interface that has been designed in an application development step. When the application is running and VisualLift detects that an update on the host screen occurred, a panel identification process is performed to find out which PWS window of the application corresponds to the new or updated host screen. This is done by inspecting the panel identification conditions of each PWS window of the VisualLifted application.

The screen of a 3270 host application is composed of a sequence of text elements called 3270 fields. A field can be a static text that cannot be modified by the user, or an input field where the user can type in data. Thus, a host screen can be regarded as a sequence of fields with individual positions, sizes, contents, and attributes. The panel identification algorithm uses the information about the positions and sizes of the fields in the currently active host screen for panel recognition. If the panel identification mode "Use all fields" has been selected at application development time, the protection state (protected/unprotected against modification) of each field is also respected. That is, the RTE finds a corresponding PWS window for the currently active host screen only if the positions, sizes, and protection states of all fields of that host screen are identical to the positions, sizes, and protection states of the host screen object that has been used at application development time to generate the panel ID. If, for example, the protection fields need not be respected as the host dynamically changes the protection state of fields, the "Use all fields, ignore protected attributes" algorithm must be selected.

The two default panel identification algorithms "Use all fields" and "Use all fields, ignore protected attributes" can be modified using "Extend/Modify panel ID algorithm". In most cases, the default panel identification modes are sufficient, but there are cases where the panel identification conditions must be modified, for example:

- There are two or more screens of a host application with an identical field structure: the panel identification cannot differentiate between those host screens. If different PWS windows have been designed to map the host screens, the application developer may differentiate between them by specifying additional panel identification data.
- Some panels of a host application have dynamic portions, that is, the structure of a host panel may change at run-time and therefore is different for different usage scenarios of the same application. The application developer may exclude the dynamic areas from panel identification to be able to map host screens with different field structures using one PWS window.

See the **online help** for “Building a Panel ID - Task 5” on page 57 for further explanation.

Application Switching

It is possible to invoke a VisualLifted application from within another VisualLifted application. This feature is called *application switching*. A reason for exploiting the application switching feature may be:

- There are different host applications that are VisualLifted as separate applications and can be invoked separately, but on the host one application is called internally by another one. For example, there may be a "print" application that can be invoked by its own, but can also be called from different other applications whenever the "print" task is needed.
- The host application consists of a large number of screens which would cause the VisualLifted application file to become very large. For performance reasons it may be useful to split up one large VisualLifted application into multiple smaller ones. See also “General Hints” on page 81.

Application switching is controlled by application switch events and is performed in the following steps:

- The currently active VisualLifted application is closed.
- The specified VisualLifted target application is opened. The invocation string that is specified for the target application is *not* sent. Instead, it is assumed that the currently active host screen is already part of the target application. This also applies to the other application invocation parameters.
- All application-specific information of the source application is invalidated, because it does not apply to the target application.
- If it is desired to establish switch-back mechanisms to return to the source application, this must be done by specifying application switch events in the application event file of the target application. There is no explicit return mechanism for application switches.
- If the application end conditions are fulfilled for a target application, the RTE ends, that means that also in this case there is no switch-back mechanism.

Mapping of Host Cursor and Input Focus

Here and on the next pages you will find lists in which the items are grouped according to priority. In these lists, item 1 has the highest priority. If item 1 does not apply, item 2 applies. If item 1 and item 2 do not apply, item 3 does, and so forth. The last item in a list applies if all other previously listed items did not apply.

The host cursor is mapped to the PWS input focus observing the following priorities:

1. If a mapping routine is defined that uses the function *FclSetFocusVT*, set the input focus to the control that is referenced by the *FclSetFocusVT* service.
2. Set the input focus to the control that is defined with the *InitialFocus=Yes* attribute.
3. Set the input focus to the control having a value mapping assignment to the host field where the host cursor is currently located.
4. Set the input focus to the first control of the PWS window.

The PWS input focus is mapped to the host cursor observing the following priorities:

1. If a mapping routine is defined that uses the function *FclMapSetCursorPosition*, set the host cursor to the specified position.
2. Set the host cursor to the position referenced by the mapping parameter of the control that is defined with the *HostFocus=Yes* attribute.
3. Set the host cursor to the position referenced by the value mapping parameter of the control that currently has the PWS input focus.
4. Set the host cursor to the field that is last modified by PWS⇒host value mapping.

Selection of the Active Notebook Page

If a PWS window contains a notebook control, the notebook page preselection is done observing the following priorities:

1. If a mapping routine is defined that uses the function *FclSetFocusVT*, preselect the notebook page containing the control that is referenced by the *FclSetFocusVT* service.
2. Pre-select the notebook page that is defined with the *InitPage=Yes* attribute.
3. Pre-select the notebook page containing the control that has a value mapping assignment to the host field where the host cursor is currently located.
4. Preselect the first notebook page.

Data Mapping

Data mapping in **host⇒PWS** direction determines which value will be used for the control in the PWS window. It is performed observing the following priorities:

1. If a mapping routine is defined, use the value that is specified for the control in that routine, using the *FclPutVarVT* or *FclInsertListRowVT* service.
2. If the access type of the control is *Out* or *InOut*, perform the host⇒PWS value mapping that is defined for the control, including padding (refer to “Padding” on page 18).
3. If there is an *Init* parameter specified for the control, use this value. The *Init* parameter has only an effect if the access type is *In*, or the access type is *InOut*, or the result of the host⇒PWS mapping that is defined for the control is an empty string.
4. Empty the value (except if no mapping is specified at all).

Data mapping in **PWS⇒host** direction determines which value will be written into the host field. It is performed observing the following priorities:

1. If a mapping routine is defined, use the value that is specified for the field in that routine, using the *FclMapUpdateField* service.
2. If the access type of the control is *In* or *InOut*, perform the PWS⇒host value mapping that is defined for the control, including padding (refer to “Padding” on page 18).
3. Do not modify the host field.

Mapping of Colors and Attributes

The PWS control **colors** are determined observing the following priorities:

1. If a mapping routine is defined, use the colors that are specified for the control in that routine, using the *FclPutVarVT*, *FclInsertListRowVT*, or *FclSetListModeVT* service.
2. If the *MapEmphasis* or *MapListEmphasis* parameter is specified for the control, use the colors that are defined there.
3. Use the colors that are statically defined using the *Color* and *BackColor* parameters.
4. Use the system default colors.

Color mapping is possible for the following controls:

- CheckBox
- PushButton (only for OS/2)
- EntryField
- MultiLine
- Slider (only for OS/2)
- SpinButtonGroup (only for OS/2)
- List
- ComboBox
- RadioButton
- Static(Text)

The PWS control **attributes** (visible/invisible, enabled/disabled, protected/unprotected) are determined observing the following priorities:

1. If a mapping routine is defined, use the attributes that are specified for the control in that routine, using the *FclPutVarVT*, *FclInsertListRowVT*, or *FclSetListModeVT* service.
2. Determine the control attributes using one of the following mapping mechanisms:
   - *EnableBy*, *DisableBy*, or *Enable=No* parameters are defined for the control that allow to set the enable/disable attribute depending on the setting of another control.
   - The protection attribute is used to define the control as being protected.
   - If the host field that is assigned to the PWS control by value mapping is protected, the PWS control also becomes protected. Controls that do not support the protected state become disabled.
   - If the *MapEmphasis* or *MapListEmphasis* parameter is specified for the control, use the attributes that are defined there.

   Multiple attribute settings are connected by a logical OR condition. For example, if a control is disabled by a *MapEmphasis* parameter, the disable state of the control will not be overwritten by an *EnableBy* parameter.
3. Use the default attribute for the control (visible, enable, unprotected).

Padding

The definition of padding characters affects the behavior of host⇒PWS as well as PWS⇒host data mapping. Up to four different padding characters may be specified, where PWS⇒host mapping uses only the first one.

When performing **host⇒PWS** mapping, all specified characters will be used to eliminate trailing padding characters from the value in the host field before it is copied to the PWS value. For example, specifying *Padding="."* will remove the last three dots from the host value ".../..." (resulting in PWS value ".../"). Specifying two padding characters, *Padding="."* and *Padding="/"*, will remove all characters from the host value (resulting in an empty PWS value).

When performing **PWS⇒host** mapping, only the first padding character will be used to fill the host field up to the available length if less characters are specified for the PWS value. For example, specifying *Padding="_"* for an eight byte field will result in "ABC_____" as host value, if "ABC" has been specified as PWS value.

Other examples for using padding characters may be:

- *Padding=" ", Padding="_"*
  For host⇒PWS mapping, this will remove trailing blanks and underscores (which are often used as initialization characters) from the host field. For PWS⇒host mapping the host field is filled up with blanks.
- *Padding=No*
  For host⇒PWS mapping the host field value is copied unchanged to the PWS value. For PWS⇒host mapping the PWS value is copied also unchanged to the host value. If the length of the specified PWS value is shorter than the length of the host value, the rest of the initial host value is not touched. For example, if the initial host value is "YELLOW" and the specified PWS value is "M" (not "M "), the resulting host value will be "MELLOW".
- *Padding=Null*
  Hexadecimal zero '00'X is used as padding character.

If no padding is specified, Padding=" " is assumed.

VisualLift and LAN

VisualLift is enabled to be used by several users on a LAN. Installing VisualLift on a LAN enables the user of a Visual*Lifted* application to work with a LAN based VisualLift. Figure 9 illustrates the principle.

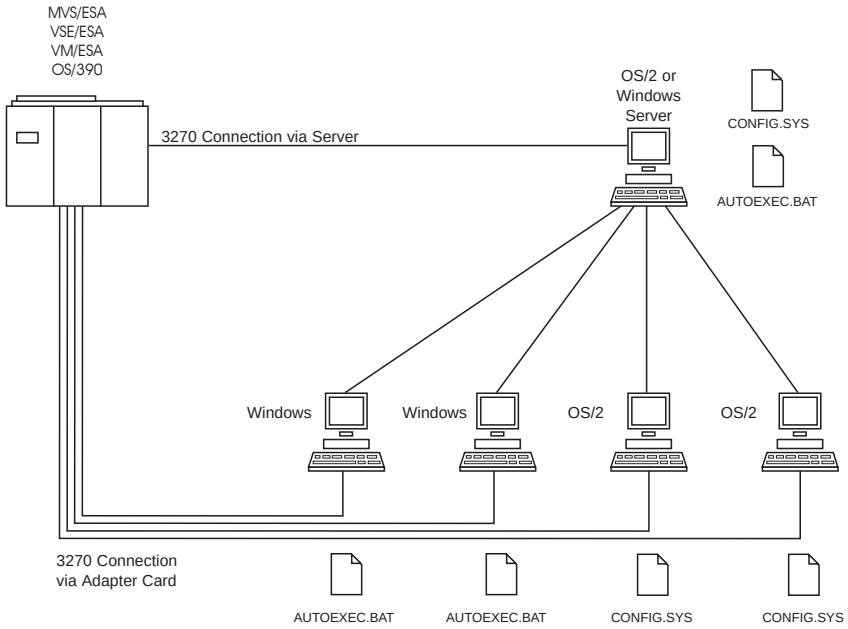


Figure 9. VisualLift and LAN

Generally observe the following rules:

- The VisualLift directory structure (fclroot/FCL) must be both available on the LAN (read-access) and on a user specific-device. This device can be
  a local device or
  a user specific LAN device where the user has write-access.
  The directories on the user-specific device are initially empty.
  If, for example, VisualLift is installed on `X:\LANLIFT` on the LAN server, and the local copy of the VisualLift root directory structure is located on `D:\LIFT`, set **FCLROOTLAN** to the LAN directory and **FCLROOT** to the local directory:

  ```
  SET FCLROOTLAN=X:\LANLIFT
  SET FCLROOT=D:\LIFT
  ```

  Note that FCLRTS must be set for OS/2, but not for Windows. VisualLift for Windows (VLW.EXE) must be started from the LAN directory. Specify `X:\LANLIFT` as working directory for the program item (icon).
- Visual*Lifted* applications may be installed on the server or on the client. VisualLift RTE first tries to find a Visual*Lifted* application on the client workstation (using the *FCLROOT* environment variable) and second on the server workstation (using the *FCLROOTLAN* environment variable).
  If you start VisualLift for Windows from the LAN without an application name, the window prompting for the application names only shows the names of those applications residing on the LAN server. Nevertheless it also works if you enter names of applications which are on the local drive.

- If VisualLift for OS/2 and VisualLift for Windows are both installed on one LAN server, the directories can reside on the same drive but must not be merged. Both require a complete directory structure and separate copies of the application files.

LAN Installation (Server)

The installation of VisualLift on a LAN server is described in Chapter 4, “Installation” on page 23.

LAN Installation (Client)

To enable a client workstation to use VisualLift from a LAN server, perform the following steps:

- Create a copy of the *fclroot* \FCL directory tree containing the following sub-directories:

  ```
  \FCL\ASDATA
  \FCL\NLSDATA\ENGLISH
  \FCL\TMP
  ```

  If application development is performed on the local workstation, the following sub-directories must also be contained:

  ```
  \FCL\USEREXIT
  \FCL\HOSTSCAN
  \FCL\PWSTEXT
  \FCL\PROJECTS
  ```

- Update CONFIG.SYS (for Windows AUTOEXEC.BAT) with the following entries:

  ```
  SET FCLROOT=D:\LIFT
  SET FCLROOTLAN=X:\LANLIFT
  ```

  for OS/2:

  ```
  SET FCLRTS=X:\LANLIFT
  ```

  add `X:\LANLIFT\RTS\DLL` to the LIBPATH statement
  add `X:\LANLIFT\RTS\BIN` to the PATH statement
  add `X:\LANLIFT\RTS\HLP` to the DPATH statement
  add `X:\LANLIFT\RTS\HLP`, `X:\LANLIFT\FCL\NLSDATA\ENGLISH`, and `D:\LIFT\NLSDATA\ENGLISH` to the HELP statement

  If you are performing application development, also update CONFIG.SYS (for Windows AUTOEXEC.BAT) with the following entries:

  for OS/2:

  ```
  SET FCLADS=X:\LANLIFT
  ```

  add `X:\LANLIFT\ADS\DLL` to the LIBPATH statement
  add `X:\LANLIFT\ADS\BIN` to the PATH statement
  add `X:\LANLIFT\ADS\HLP` to the DPATH statement
  Also add `X:\LANLIFT\FCL\USEREXIT` to the LIB and INCLUDE statement.

  Optionally you can use the environment variables described at “Environment Variables” on page 101.

Host Connection

Even when the client workstation is connected to a LAN, it still needs a host connection - either via the LAN or via an adapter card installed in the client workstation.

National Language Support

VisualLift is available in English, only. However VisualLift OS/2 is enabled to provide national language support for Visual*Lifted* applications for single- and double-byte character languages. Even if the host application is available, for example in English only, the Visual*Lifted* user interface can be translated into any single- or double-byte character language.

The solution: when the VisualLift user interface is developed, the NLS part of the user interface is stored in separate text files. These text files can be translated. The window layout of the Visual*Lifted* application adapts automatically to the translated text.

The NLS support is enabled for windows, controls, and helps. This means, that the text files for windows, controls, and helps may be translated. At run-time the VisualLift windows dynamically accommodate the text lengths and the appropriate controls are displayed.

VisualLift Windows RTE supports single-byte character languages.

Adding Your Own Functions to VisualLift

The set of functions offered by the RTE can be extended by writing application provided routines. VisualLift provides exits to include the application provided routines. This enables for:

- Extensive input validation on the workstation
- Integration of other, workstation based, applications
- Access to and modification of the 3270 data stream
- Dynamic exchange of help information
- Integration of user defined controls, for example business graphics.

How to write application provided routines is described in the VisualLift Reference. The following samples of application provided routines are available:

- Checking routines
- Mapping routines
- Action routines
- Help exit routines
- Window procedures for private controls
- Panel identification exit routines.

For more information concerning application provided routines refer to “Supplying Application Provided Routines” on page 67. Samples of application provided routines are available in C-source code in the file SAMPLE.C in directory *fclroot*\FCL\USEREXIT.

Chapter 4. Installation

VisualLift runs under OS/2 and Windows and is delivered on:

- Tapes for MVS to download VisualLift from MVS
- Tapes for VSE to download VisualLift from VSE
- Tapes for VM to download VisualLift from VM
- Diskettes to install VisualLift directly on the workstation.

The procedures to install VisualLift on the host (MVS,VSE, or VM) are described in the following manuals:

- VisualLift Program Directory for MVS
- VisualLift Program Directory for VSE
- VisualLift Program Directory for VM.

VisualLift Run-Time Environment (RTE) for OS/2 and VisualLift Run-Time Environment (RTE) for Windows are part of OS/390. To install VisualLift RTE, you have to download it from the host where you installed OS/390.

Host Installation Process

The VisualLift host installation process consists of two steps:

1. Read the installation description file FCLREADM on the host. The installation description of VisualLift is available
   - for MVS: as a member in the partitioned data set `'SYS1.SFCLINST.SFCLINST(FCLREADM)'`
   - for VSE: as member FCLREADM W in PRD2.PWS
   - For VM: as FCLREADM DES *

   To read the installation description file, download it to your workstation in binary format.
2. Perform the down-load of VisualLift from the host to the workstation. Follow the instructions in the `FCLREADM` file.

Diskette Installation Process

Read the installation description file `A:\FCLREADM.TXT` on Diskette 1 and follow the instructions.

Hard Disk Space

To install VisualLift ADE and RTE on your workstation approximately 10MB (including 4MB temporary for installation) of disk space are needed.

To install VisualLift RTE (either OS/2 or Windows) on your workstation approximately 6MB (including 2MB temporary for installation) of disk space are needed.

System Requirements

Before you start the installation procedure, make sure that the workstation where VisualLift is to be installed fulfills the system requirements as described in Appendix A, “System Requirements” on page 97.

Different Levels of VisualLift

This section is only applicable if you already installed VisualLift (either VisualLift RTE or VisualLift ADE) on your workstation or on a LAN. Typical situations are:

- You already developed Visual*Lifted* applications on this workstation or on the LAN - using VisualLift ADE.
- You are a user of VSE Workdesk.
  If the VSE Workdesk (part of VSE/ESA 2.x) is already installed on the workstation where you want to install VisualLift RTE, you must install the VisualLift RTE in the same drive and path as the VSE Workdesk.

In general, the various versions of VisualLift are upward compatible. This means, the most recent level of VisualLift supports the entire functionality of the previous levels of VisualLift. Make sure that you replace an existing VisualLift only with a more recent level of VisualLift.

Check Service Levels

You already used VisualLift RTE on the workstation or on the LAN. If you receive another VisualLift RTE you have to know which level is the most recent one. To check the service level of VisualLift RTE do the following:

- View the file `FCL.DAT` in the *fclroot* directory.
- View the service level information in the FCLREADM installation description file.
- View the information delivered with a Visual*Lifted* application.

The higher number indicates the more recent level of VisualLift RTE. Use the most recent level of VisualLift RTE.

Chapter 5. Getting Started

After installing VisualLift the VisualLift folder resides on the desktop. The VisualLift folder contains a set of objects which are described in this chapter. Double-click on the VisualLift icon to open the folder.

Object Hierarchy

The following figure shows the hierarchy of VisualLift objects. It is assumed that the sample applications and the Solution Guide are installed.

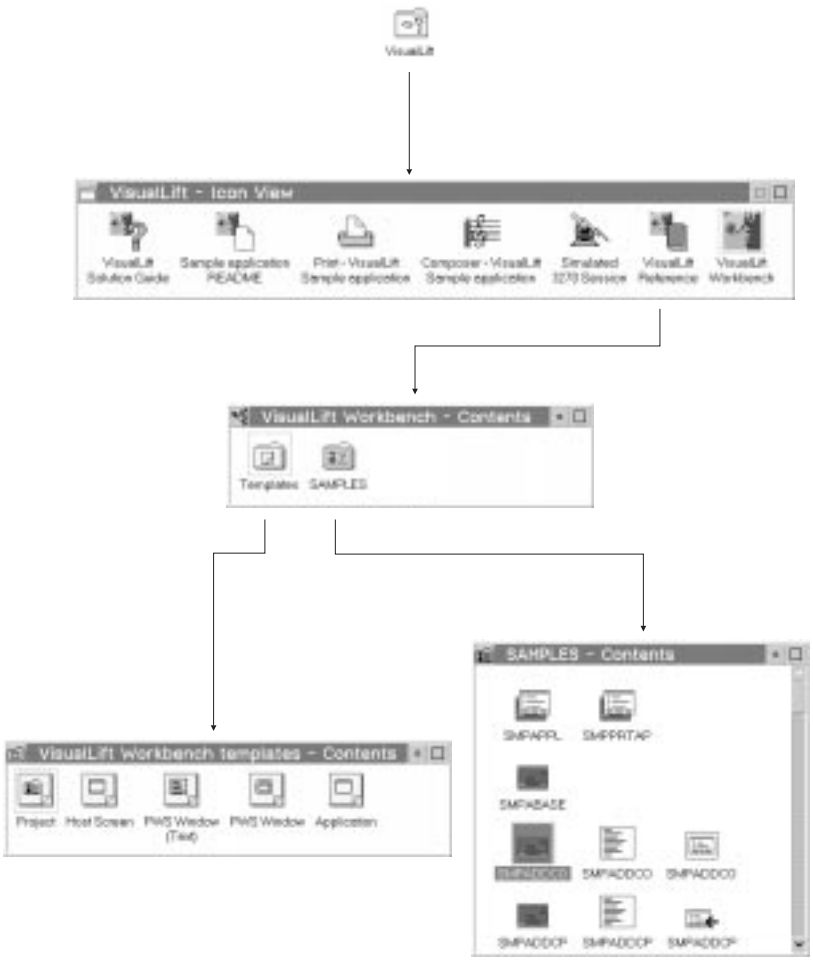


Figure 10. Hierarchy of VisualLift Objects

Simulated 3270 Session

The simulated 3270 session simulates a 'real' 3270 host session. It looks and behaves similiar to a 3270 emulator window, but is only used within VisualLift for demonstration purposes. The sample applications as well as the Solution Guide use the simulated 3270 session to show the run-time behavior of VisualLift applications. You must start it **before** you start one of the sample applications or the Solution Guide.

The simulated 3270 session can also be used without the sample applications or Solution Guide to simulate the flow through a sequence of different host screens. The **Sample Application README** describes what you can do to initiate and control this flow through the host screens.

Composer - VisualLift Sample Application

The composer application is one of the two applications provided by VisualLift to show the run-time behavior of a VisualLifted application. Instead of running against a 'real' host session the sample applications are running against the simulated 3270 session. They give examples of how host fields can be mapped to workstation controls, and show all features provided by VisualLift such as events, online helps, or application provided routines.

Print - VisualLift Sample Application

This is the second sample application provided by VisualLift. It can be performed on its own or can be called by another application via application switch.

Sample Application README file

The sample application README file describes all possible interactions for using the objects

- Simulated 3270 session
- Composer - VisualLift Sample Application
- Print - VisualLift Sample application

VisualLift Workbench

The VisualLift Workbench is used to develop a VisualLift application. Chapter 6, “Using VisualLift” on page 31 describes in detail the necessary steps to Visual*Lift* a host application. It also shows the purpose and use of the objects contained in the workbench.

Important

This chapter is **highly** recommended for novice users

VisualLift Online Solution Guide

The Online Solution Guide offers guidelines for the design of workstation windows, and shows the usage of each workstation control for your reference.

The quality of a Visual*Lifted* application depends on the design of its workstation windows which decides about its usability and ease of use.

Hint

The Solution Guide is **the** tool which helps you to find the corresponding workstation control for a host field.

Design Guidelines

The design guidelines help VisualLift users to design workstation windows which take as much advantage of the workstation user interface as possible. The following list provides recommendations concerning the design of workstation windows.

- Study and analyze the host screen: what are the semantics of each input/output field, what is the context of the host screen?
- Search for the most intuitive workstation control for the host input/output field.
- Is there a way to group several input/output fields into one workstation control?
- If more than one workstation control is eligible for the host input/output field, choose the control which is intuitive for the user of the Visual*Lifted* application.
- Identify each workstation control or group of controls with a prompt text, a column heading, or a window title, whichever is most appropriate.
- Use a prompt text that clearly indicates the function of a workstation control.
- If VisualLift provides a workstation control that supplies the required function, use that control rather than creating application provided routines.

Accessing the Solution Guide

Make sure you have already started the simulated 3270 session.

1. Double-click on the ***Solution Guide*** icon

2. The *VisualLift Solution Guide - Main selection* window appears:

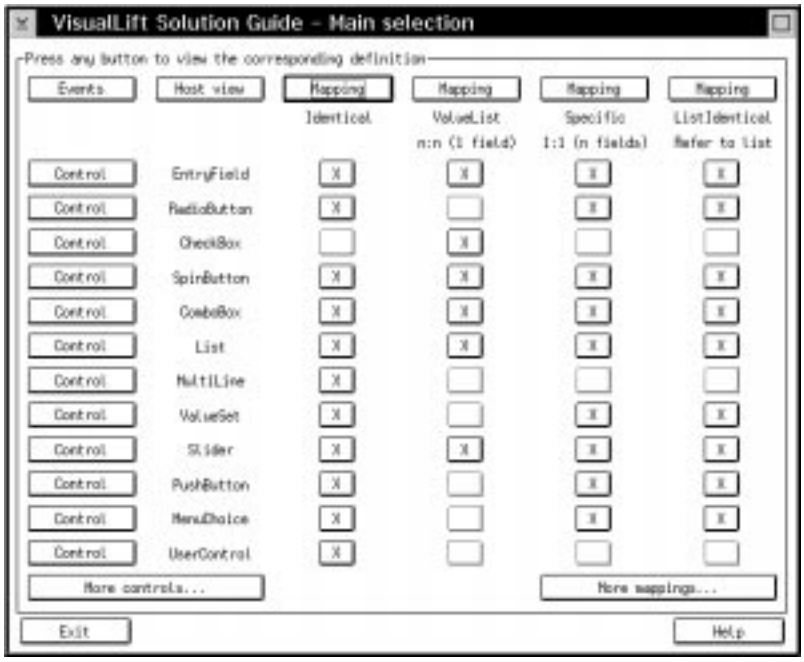


Figure 11. VisualLift Solution Guide - Main selection

Controls - Definitions and Examples

The left side of the window shows a column of controls.
Clicking on the ***Control*** push button to the left of a listed control results in a window with the definition of the control and examples what it may look like. The following window appears when you click on the *Control* push button to the left of *EntryField*.

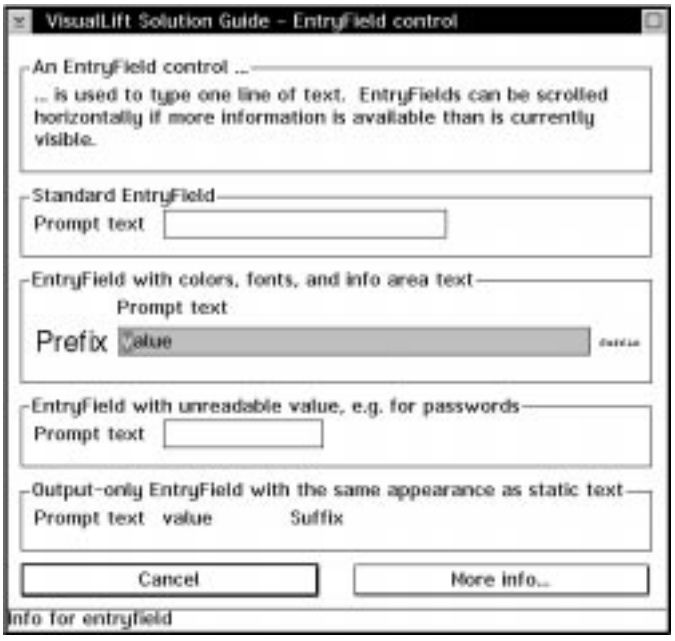


Figure 12. VisualLift Solution Guide - EntryField control

More controls... at the bottom of the *Main selection* window shows additional controls. Clicking on any of the ***Control*** push buttons also shows the definition of the selected control and shows an example what it may look like.

Mapping Modes - Information and Explanation

At the top of the *Main selection* window is a row of push buttons showing various MAPPING modes. Clicking on any of the *Mapping* push buttons provides you with information about the selected mapping mode.
Clicking on ***More mappings...*** at the bottom of the window shows additional mapping related examples.

Controls and Mapping - Information and Explanation

Clicking on one of the buttons with an **X** shows you how a value on the workstation window is mapped to the host panel by using a specific combination of a control and one of the mapping modes. The following example is for *EntryField and Identical Mapping*:

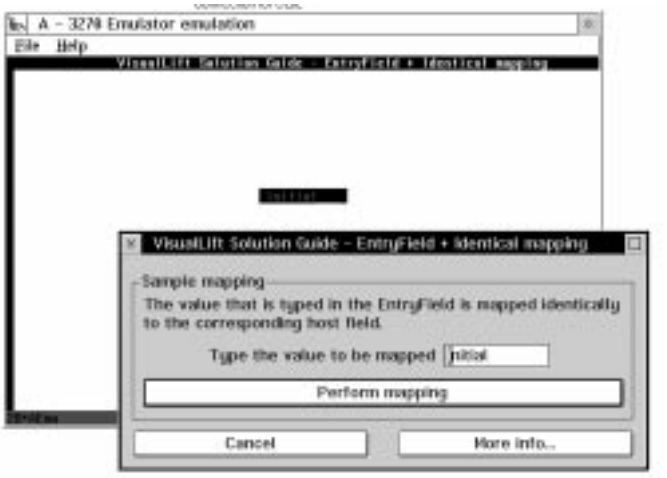


Figure 13. VisualLift EntryField+ Identical Mapping

Host view shows parts of host situations and leads to the suggested workstation solution.

Events explains in detail what an event is and shows examples for different types of events.

More info - Information and Explanation

More info ... is given for controls and mapping modes.
More info... for *controls* provides:

- An explanation of the control and advice where to use it.
- Information which mapping mode can be used together with the selected control.
- Information where the definition of the selected control can be found in the VisualLift Online Reference.
- Information where the selected control is used in the sample application.
- A description how to define the selected control with the graphic editor.
- A description how to define the selected control using the textual approach.

More info... for *mapping modes* provides:

- An explanation of the selected mapping mode.
- A reference to the corresponding definition of the selected mapping mode in the VisualLift Online Reference.
- Information where the selected mapping mode is used in the sample application.
- A description how to define the selected mapping mode using the graphic editor.
- A description how to define the selected mapping mode using the textual representation.
- The name of the host screen object currently visible in the simulated 3270 session window.

Chapter 6. Using VisualLift

This chapter describes how to Visual*Lift* a host application. Visual*Lifting* a host application consists of several tasks. The tasks are shown in Figure 15 on page 33.

You create a project named *COMPOSER* which becomes part of the VisualLift Workbench and is used as a base for examples throughout this chapter.

It is recommended especially for novice users to perform *all* tasks to get a basic understanding what Visual*Lifting* is all about.

Terminology

While working with VisualLift some expressions or terms may be new to you. For example, the term *application* within VisualLift is the Visual*Lifted* user interface for a host application. Whenever an unknown term occurs, do one of the following:

- refer to “Glossary” on page 105
- use the index of either the VisualLift Help Facility or Reference or this user's guide
- use the VisualLift Help Facility or Reference; also use the hypertext links provided there.

VisualLift Tasks

The VisualLift tasks described on the next pages provide you with a framework how to Visual*Lift* a host application. The principles of working with VisualLift as well as some reductions for your work are provided.

Figure 15 on page 33 shows the sequence of tasks and gives a short overview what happens within each task. For novice users the sequence of tasks is a **must**.

Start

Within that section of the document you will receive exact guidance how to proceed. Each task will be described in a structured way. For your orientation each page identifies the task you are actually performing by a running heading at the top of the page. Special emphasis is given to remarks, hints and recommendations.

In the task section the following host screen is the model to create a Visual*Lifted* window.

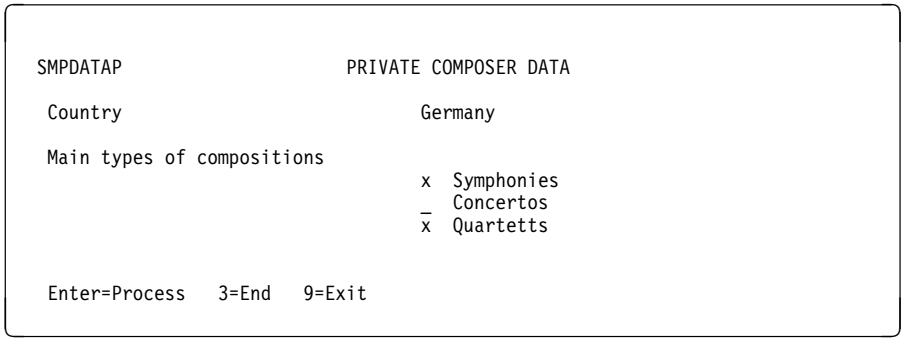


Figure 14. The Host Screen Model to Create a VisualLifted Window

Interaction Techniques

VisualLift's ADE runs on OS/2 and the interaction is object oriented. If you are not familiar with these techniques, it is recommended to study these techniques either by using the OS/2 tutorial or reading OS/2 documentation. To use VisualLift, knowledge of the following concepts and interaction techniques is required:

- Folder concept
- Template concept
- The use of context menus for objects
- The use of context menus for folders
- Drag and drop techniques.

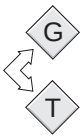
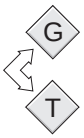
VisualLift Set-up	Task 0		Set-up the host: navigate to the host application to be VisualLifted, but do not start the host application. Set-up VisualLift: within the settings notebook enter the characteristics of the VisualLift Workbench.
Create Project	Task 1		Create a project within the VisualLift Workbench. Data belonging to that project will be created and manipulated within that project. Create one project for every host application to be VisualLifted.
Scan Host Screen	Task 2		Navigate to the first screen of the host application. Then create the host screen object. The scanned host screen is the base for VisualLift. View a scanned host screen by a double-click on the icon.
Design PWS Window	Task 3		This is the most important step of VisualLifting. The quality of the VisualLifted user interface depends on the transformation of host input/output fields into workstation controls.
Perform Mapping	Task 4		Once the workstation control elements are designed they have to be mapped to the corresponding host input/output fields. This task is to be performed for each individual workstation control.
Build Panel ID	Task 5		Each VisualLifted window has to be assigned to a host screen. The VisualLifted window has to be unique within an application (or project). Therefore an identifier for each window has to be created.
Build Application	Task 6		All VisualLifted windows are linked together to build the VisualLifted application. Additionally the invocation and termination may be defined.
Test VisualLifted Application	Task 7		This is the first time a VisualLifted application works in conjunction with the host application. At that point the VisualLift RTE starts to work.
Distribute VisualLifted Application	Task 8		Distribute the VisualLifted application and copy it to a directory where it is accessed by the VisualLift RTE. The distribution of the VisualLifted application can be done via diskettes, via LAN, or via the host.
Run VisualLifted Application	Task 9		Associate the VisualLift RTE object with the VisualLifted application. Start the VisualLifted application by a double-click on the application icon.

Figure 15. Tasks to VisualLift a Host Application

Setting-up VisualLift - Task 0

It is assumed that the VisualLift installation is finished. VisualLift requires a defined environment to work correctly. Perform the host environment set-up either for MVS, VSE, VM or OS/390.

Setting-up Your Host Environment

Open one of your host sessions.

1. Log on to your User-ID
2. Go to the screen where you can start your host application, but do not invoke it yet.

Important

In this chapter the 3270 emulator simulation provided by VisualLift is used instead of a 'real' host session.
If you are performing the tasks described in this chapter, setting-up your host environment is equivalent to setting-up the emulator simulation.
Double-click on the **simulated 3270 session** icon in the VisualLift folder.

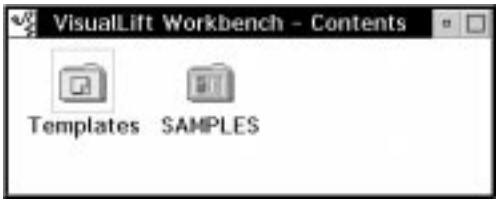
You have successfully established your working environment.

Setting-up Your VisualLift Environment

Start VisualLift.

1. Double-click on the ***VisualLift Workbench*** icon in the VisualLift folder

The VisualLift Workbench is located in the VisualLift folder on the desktop.

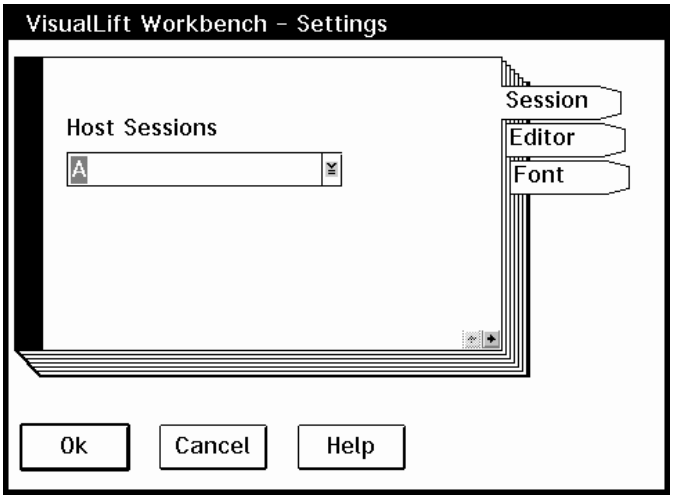


2. Click with mouse button 2 on the background of the *VisualLift Workbench - Contents* folder
3. Select in the context menu ***Open as settings***

The *VisualLift Workbench - Settings* notebook appears.

Enter the settings of VisualLift.

1. Enter the identifier of the ***Host session***
2. Select the ***Editor*** tabbed section
3. Enter the name (including drive and path) of the text ***Editor*** you want to use
4. Select the ***Font*** tabbed section
5. Select the *Font*
6. Press the ***OK*** push button of the notebook



Creating a Project - Task 1

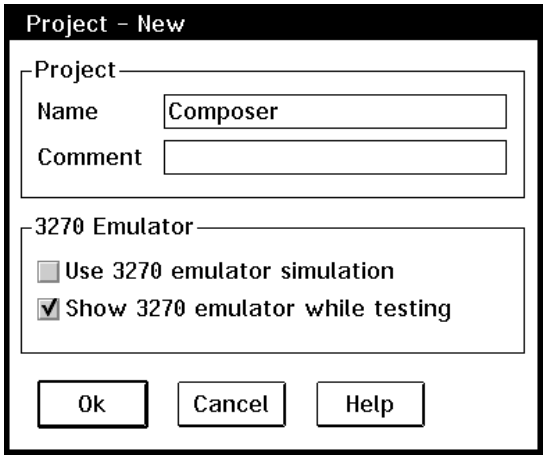
Before you Visual*Lift* an application you have to create a project. A project is the area where the actual VisualLifting takes place.

The work during Visual*Lifting* will take place in project folders. It is recommended to maintain all data belonging to *one* Visual*Lifted* application within *one* project folder.

You can either define a project using the **context menu** or using **drag and drop**.

Define project (named: ***Composer***) using the context menu.

1. Click with mouse button 2 on the background of the *VisualLift Workbench - Contents* folder
2. Select in the context menu ***Create project***
3. Enter ***Name*** and ***Comment*** (optional) in the *Project - New* window.
4. Click on the first check box for *3270 Emulator*. This selection enables testing of the *VisualLifted* window against the *Simulated 3270 session* of the *VisualLift Sample application*. The second one has been selected by default.
5. Press the ***Ok*** push button in the window



6. Result: The *Composer* project is created

Important

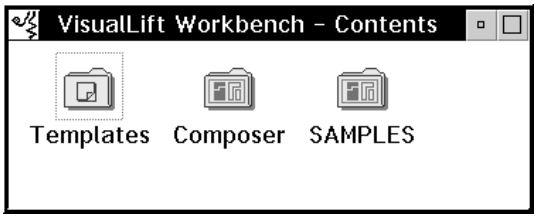
Select the first check box only when performing the following tasks, **not** when you are lifting your own applications.
The second check box can stay selected. You can see the host session simultaneously to the lifted application, and check their relationship while testing.

Define project (named: ***Composer***) via drag and drop.

1. Double-click on ***Template*** icon
2. Drag the ***Project*** icon in the *Template* folder and drop it over the *VisualLift Workbench - Contents* folder
3. Enter ***Name*** and ***Comment*** (optional) in the *Project - New* dialog box
4. Press the ***OK*** push button in the window
5. Result: The *Composer* project is created

If you want to create other objects also using the templates, leave the Template folder open.

The project ***Composer*** is created and visible in the *VisualLift Workbench - Contents* window.



Scanning a Host Screen - Task 2

Now you have to capture a host screen of a host application. This host screen provides the base from which the workstation window is designed (see “Designing a PWS Window - Task 3” on page 40). Navigate to the screen to be captured in your host session, the simulated 3270 session in this case.

Enter in the *Simulated 3270 session*:

1. ***composer*** and press ***Enter***
2. Enter ***3*** in the *Specify desired option* field and press ***Enter***
3. Enter ***2*** in one of the *Option* fields and press ***Enter***
4. Result: you get the *Private Composer Data* host panel displayed

This panel is scanned in this task and is the base of the VisualLifted window which will be created within the following tasks.

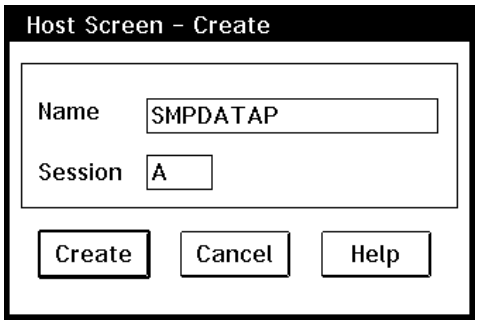
```
SMPDATAP                    PRIVATE COMPOSER DATA

 Country                           Germany

 Main types of compositions
                                   x  Symphonies
                                   _  Concertos
                                   x  Quartets


 Enter=Process   3=End   9=Exit
```

Figure 16. The Host Screen Model to Create a PWS Window

Open the *Composer* project by double-clicking on the *Composer* icon. You can either scan a host screen using the context menu or using drag and drop.

Scan host screens using context menus.

1. Click with mouse button 2 on the background of the *Composer - Contents* folder
2. Select in the context menu ***Create object***
3. Select in the cascaded menu ***Host Screen***
4. Enter ***Name*** and ***Session*** in the *Host Screen - Create* window
5. Press the ***Create*** push button in the window

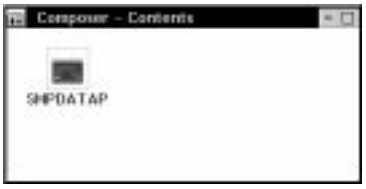


6. Result: The *Host Screen* object is created

Scan host screens using drag and drop.

1. Drag the ***HostScreen*** icon of the *Template* folder and drop it over the *Composer - Contents* folder
2. Enter ***Name*** and ***Session*** in the *Host Screen - Create* window
3. Press the ***Create*** push button in the window
4. Result: The *host screen* object is created

The host screen ***SMPDATAP*** is created and visible in the *Composer - Contents* folder.



Double-click with mouse button 1 on the host screen SMPDATAP in the COMPOSER project. The host screen is then always visible while you are defining the workstation window.

By selecting the fields you can see the field structure of the host screen. The status line at the bottom of the screen shows you the attributes of the field where the mouse pointer is currently located.

You may now close the *Simulated 3270 session*.

Designing a PWS Window - Task 3

Note: This task is the most important one! It decides about the quality of the graphical user interface you are creating for a host application. Working through the online VisualLift Solution Guide **before** you start designing PWS windows for your own applications provides you with the necessary guidelines and helps.

For the scanned host screen SMPDATAP the corresponding PWS Window has to be designed. This task has the following sub-tasks:

- Creating a PWS Window
- Defining controls in graphical mode
- Defining controls in textual mode
- Transforming graphical representation into textual representation and vice versa.

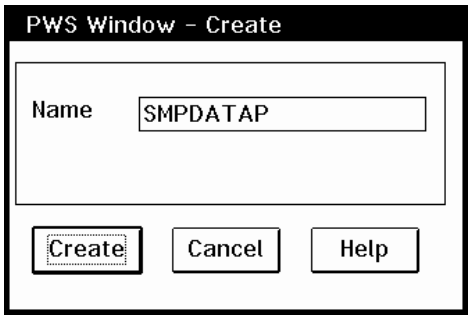
Important

The Host Screen, PWS Window, and PWS Window (Text) objects must have identical names to indicate that these objects belong together. You can distinguish them by their representation, the icon.

Creating a PWS Window

Create a *PWS Window* using context menus.

1. Click with mouse button 2 on the background of the *COMPOSER - Contents* folder
2. Select in the context menu ***Create object***
3. Select in the cascaded menu ***PWS Window***
4. Enter ***Name*** *(SMPDATAP)* in the *PWS Window - Create* window
5. Press the ***Create*** push button window

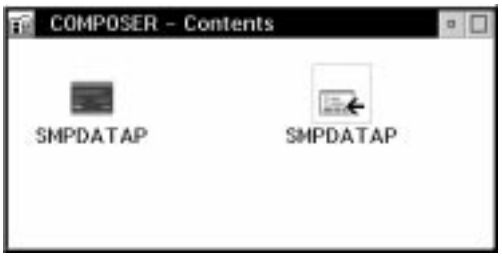


6. Result: The *PWS Window* is created

Create a *PWS Window* using drag and drop.

1. Drag the ***PWS Window*** icon of the *Template* folder and drop it over the *COMPOSER - Contents* folder
2. Enter ***Name*** *(SMPDATAP)* in the *PWS Window - Create* window
3. Press the ***Create*** push button in the window
4. Result: The *PWS Window* is created

The created *PWS Window* object appears in the *COMPOSER - Contents* folder.



Defining Controls in Graphical Mode

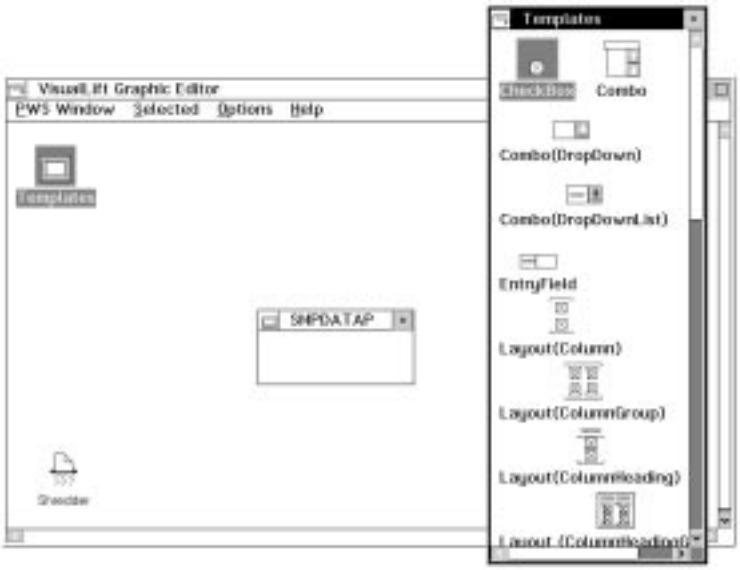
Define the workstation controls for the *PWS Window* SMPDATAP. This step is the important one. Take your time to figure out how the host input/output fields should be converted into workstation controls. For more information see the **online** VisualLift *Solution Guide*.

The following workstation controls will build up the *PWS Window*:

- one window
- one entry field for the country name
- one group box to contain the type of compositions
- three check boxes within the group box (one check box for each type of compositions)
- three push buttons for the three attention keys mentioned on the host screens.

The created *PWS Window* is empty. Create the workstation controls for the *PWS Window*.

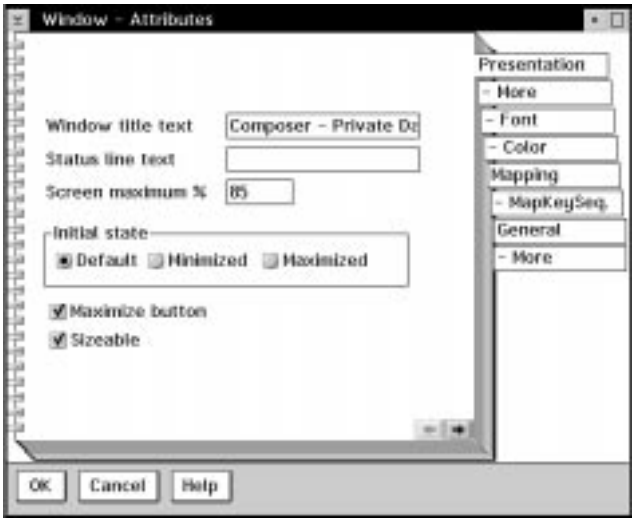
1. Double-click with mouse button 1 on the *PWS Window SMPDATAP* in the COMPOSER project



2. Result: The *graphic editor* appears

Double-click into an empty space of **SMPDATAP** window.

1. Enter *Composer - Private Data* in the ***Window title text*** field
2. Press ***Ok*** push button



3. Result: The window title is defined.
4. Click on the *S* at the left of the window title to save the window.

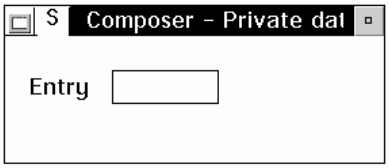
Tip

Saving the PWS window after each modification allows you to see it in its latest state using the *Play* option. See the context menu of the PWS window object SMPDATAP in the *Composer - Contents* folder for the *Play* option.

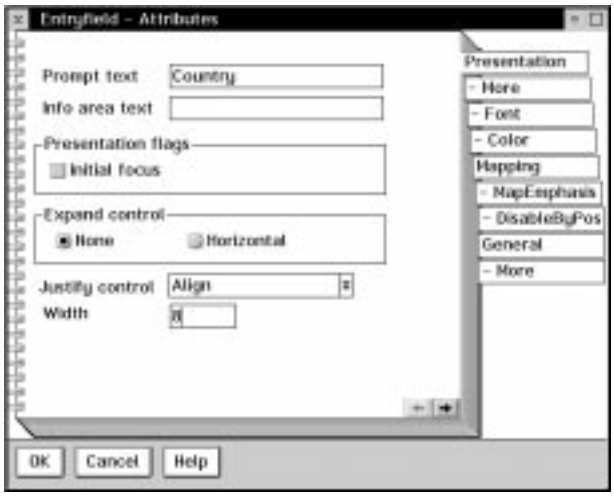
Define now the *Entry field* control.

1. Search in the *Template* window of the *graphic editor* for the control ***Entry Field***
2. Drag the ***Entry Field*** icon and drop it over the *Composer - Private Data* window.

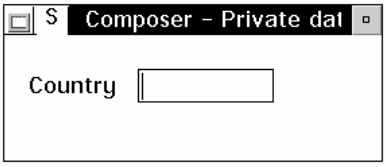
   If you want to delete a control out of your window, drag it to the shredder.



3. Result: ***Entry Field*** appears in the *Composer - Private Data* window
4. Click on the *S* at the left of the window title to save the window.
5. Double-click on the ***Entry Field*** in the *Composer - Private Data* Window



6. Result: The *Entryfield - Attributes* notebook appears
7. Enter *Country* in the ***Prompt text*** field
8. Change ***Width*** to 8
9. Press ***OK*** push button



10. Result: The entry field attributes are defined.

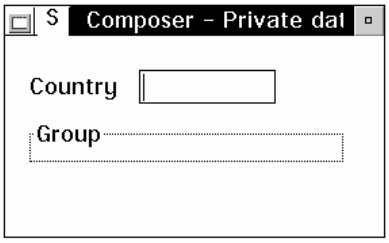
The entry field is defined and appears in the graphic editor. Click on the *S* at the left of the window title to save the window.

Tip

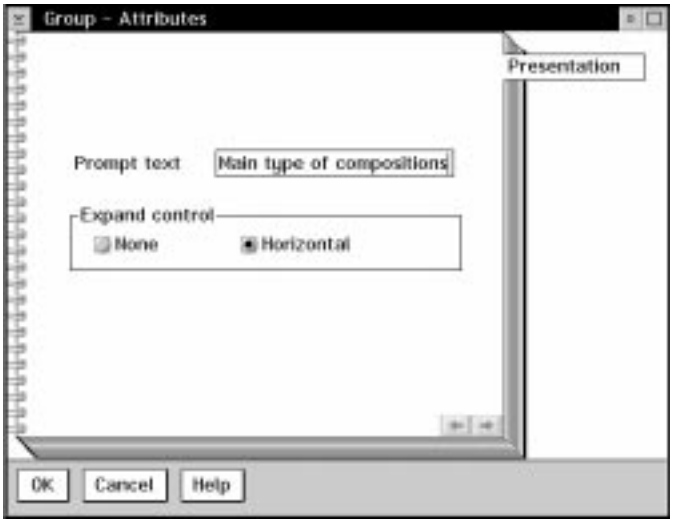
You can also drag the text for input fields from scanned host screens and drop it over the prompt text field of the notebook. This technique eliminates typing errors. See the description on page 44 how to do this.

Define now the **group box** control.

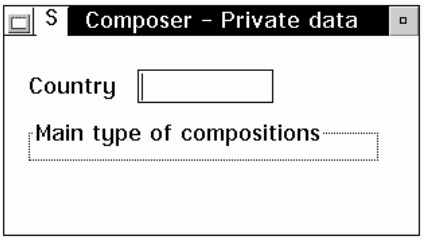
1. Search in the *Template* for the control ***Static (GroupBox)***
2. Drag the ***Static (GroupBox)*** icon and drop it over the *Composer - Private Data* window below the Country entry field



3. Result: ***Static (GroupBox)*** appears in the *Composer - Private Data* window
4. Double-click on the ***Static (GroupBox)*** in the *Composer - Private Data* Window



5. Result: The *Group -Attributes* notebook appears
6. Go to the SMPDATAP host screen and click with mouse button 1 on the *Main type of compositions* string. The field changes its selection state and becomes grey.
7. Drag the selected field and drop it over the ***Prompt text*** field
8. Press ***OK*** push button

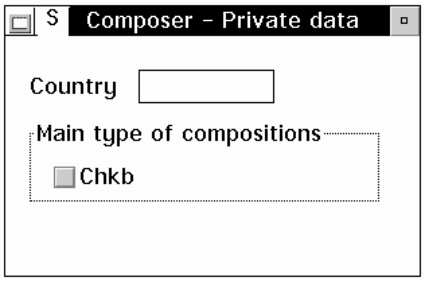


9. Result: The text of the group box is defined
10. Click on the *S* at the left of the window title to save the window.

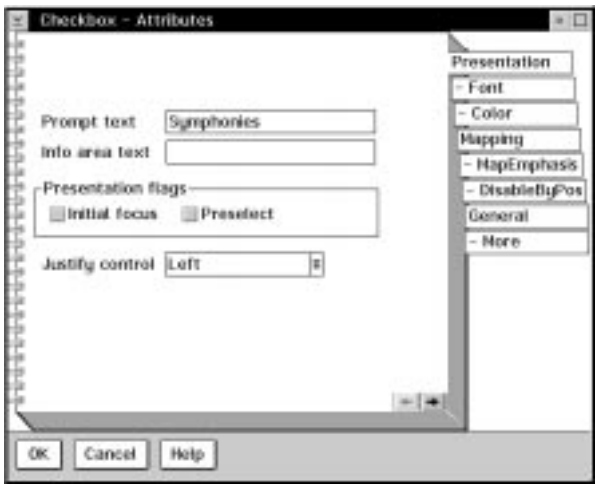
The group box is defined and appears in the graphic editor.

Define now the **check box** control.

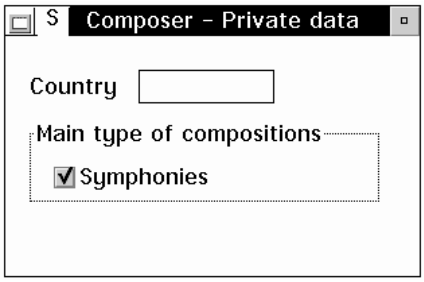
1. Search in the *Template* window of the *graphic editor* for the control ***Check Box***
2. Drag the ***Check Box*** icon and drop it over the *Main type of compositions* group box



3. Result: ***Check Box*** appears in the *Main types of compositions* group box
4. Double-click on the ***Check Box*** in the *Main type of compositions* group box



5. Result: The *Check Box - Attributes* notebook appears
6. Enter *Symphonies* in the ***Prompt text*** field
7. Press ***OK*** push button



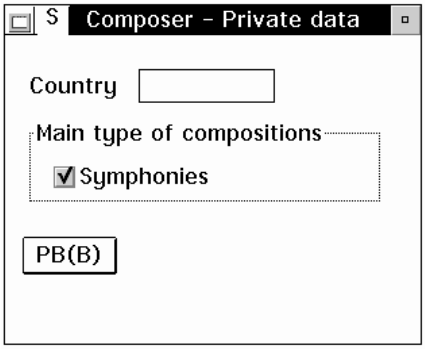
8. Result: The check box is now defined

The check box is defined and appears in the graphic editor. Click on the *S* at the left of the window title to save the window.

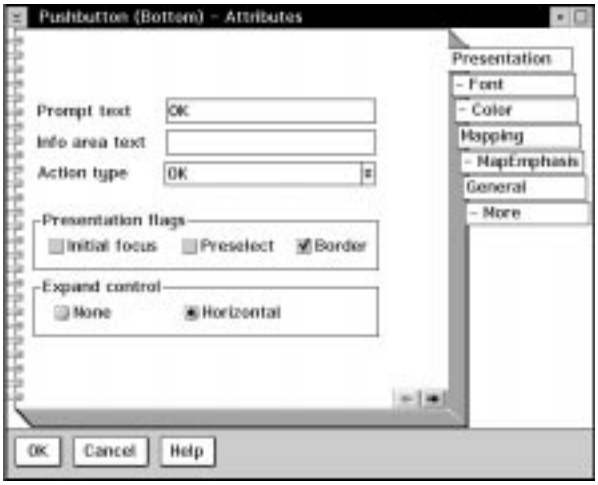
You can now repeat these steps to define the missing two check boxes, or you can define them in textual mode as described on page 48.

Define now the **push button** control.

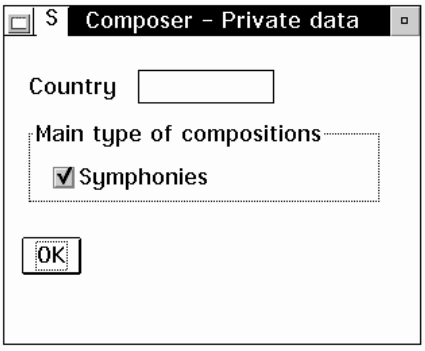
1. Search in the *Template* window of the *graphic editor* for the control ***PushButton (Bottom)***
2. Drag the ***PushButton (Bottom)*** icon and drop it over the *Composer - Private Data* window



3. Result: ***Push Button*** appears in the *Composer - Private Data* group box
4. Double-click on the ***PushButton (Bottom)*** in the *Composer - Private Data* group box



5. Result: The *Pushbutton (Bottom) - Attributes* notebook appears
6. Enter *OK* in the ***Prompt text*** field
7. Select *Horizontal* to achieve the same width for all push buttons
8. Press ***OK*** push button



9. Result: The push button is now defined

The push button is defined and appears in the graphic editor.
Click on the *S* at the left of the window title to save the window.

You can now repeat these steps to define the missing two push buttons, or you can define them in textual mode as described on page 48.

Transforming Graphical into Textual Representation

Up to now you created the main elements of a PWS Window. If you have not defined them using the *graphic editor*, there are two check boxes and two push buttons missing - they will be created in textual mode within the next step.

Within this step you will again save the created PWS Window, transform it into textual representation and define the missing controls in textual mode.“Graphical and Textual Mode” on page 9 gives an overview of textual mode and graphical mode and provides hints when to use which mode.

File the created controls and the window within the *graphic editor*.

1. Select the menu item ***PWS Window***
2. Select the pull down ***File***
3. Minimize the graphic editor

4. Result: The updates that you have made to *PWS Window* (SPMDATAP) within the graphic editor are saved to disk.

Be aware

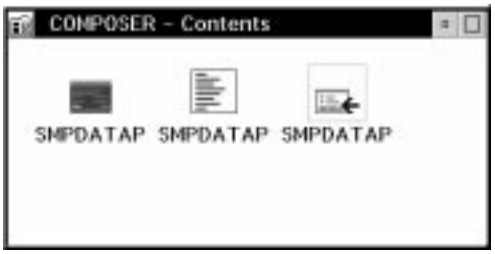
Always use the *File* action before switching to textual mode. This ensures that textual changes are visible when you invoke the graphic editor next time.

Create now a *PWS Window (Text)* object from the *PWS Window* object SMPDATAP. This process is called *transformation*. For more information refer to “Transformation” on page 82.

Create *PWS Window (Text)* SMPDATAP.

1. Click with mouse button 2 on the *PWS Window* SMPDATAP
2. Select in the context menu ***Build Text***
3. Result: The *PWS Window (Text)* SMPDATAP is created.

The created *PWS Window (Text)* SMPDATAP appears in the *COMPOSER - Contents* folder.

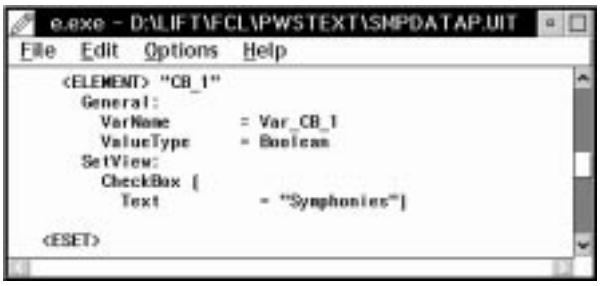


Defining Controls in Textual Mode

The missing check boxes and push buttons can be defined in textual mode.

Create controls in textual mode.

1. Double-click on *PWS Window (Text)* SMPDATAP



Result: The text editor that you defined in the workbench settings is used to edit the textual representation of the controls.

The text elements which make up a *Check Box* control look like the following:

```
<ELEMENT> "CB_1"
      General:
        VarName       = Var_CB_1
        ValueType     = Boolean
      SetView:
        CheckBox (
          Text          = "Symphonies")
```

Figure 17. Textual representation of check box 'Symphonies'

The text elements which make up a *Push Button* control look like the following:

```
<ALT> "FKA_1"
       General:
         Value         = "FKA_1"
       SetView:
         PushButton (
           Expand         =Horizontal
           Text          = "OK")
```

Figure 18. Textual representation of push button 'OK'

Note: The SetView section contains the parameters defined in the presentation part of the *Attribute* notebooks, whereas the General section contains the parameters defined in the general part.

2. Copy the *CheckBox* and the *PushButton* definition two times and place them below the original definitions.

   Note that for the added check boxes not only the text *Symphonies* must be changed to *Concertos* and *Quartetts* respectively, but also **VarName** must be changed to *Var_CB_2* and *Var_CB_3* respectively. Change the two added ELEMENT names *CB_1* to *CB_2* and *CB_3* respectively. Also for the added push buttons not only the text *OK* must be changed to *Cancel* and *Exit* respectively, but also the added occurrences of **FKA_1** must be changed to *FKA_2* and *FKA_3* respectively. Furthermore, *ActionType=Cancel* must be added for *Cancel* and *Exit*.

   In addition, references to the new controls must be added to the layout parameter at the top of the file:

- Ref= "CB_2"
- Ref= "CB_3"
- Ref= "FKA_2"
- Ref= "FKA_3"

See Figure 19 for all changes.

3.

Be aware

Save the *PWS Window (Text)* SMPDATA and close the text editor before switching to graphical mode.

4. Result: The *PWS Window (Text)* SMPDATAP is updated with the additional controls

Hint

The tags to define a PWS Window (Text) on a textual base are listed in the online VisualLift Reference. Other examples of PWS Window (Text) objects are available in the SAMPLES project folder.

```
<SET> "SMPDATAP"
  SetView:
    Window (
      Text          = "Composer - Private Data")
    Layout (
      Ref = "EF_1"
      Group (
            RefBox        = "GRP_1"
            Expand        = None)
        Ref = "CB_1"
        Ref = "CB_2"  <------------------------/ references to new
        Ref = "CB_3"  <------------------------/ controls are added
      EGroup
      Ref = "FKA_1"
      Ref = "FKA_2"   <------------------------/ references to new
      Ref = "FKA_3")  <------------------------/ controls are added
  <ELEMENT> "EF_1"
    General:
      VarName       = Var_EF_1
      ValueLength   = 20
    SetView:
      EntryField (
        Text          = "Country"
        Width         = 8)
  <SET> "GRP_1"
    SetView:
      Static (
        Text          = "Main type of compositions"
        Type          = GroupBox)
    <ELEMENT> "CB_1"
      General:
        VarName       = Var_CB_1
        ValueType     = Boolean
      SetView:
        CheckBox (
          Text          = "Symphonies")
/*************************************************************************/
    <ELEMENT> "CB_2"                          / Copied and then      /
      General:                                / modified             /
        VarName       = Var_CB_2              / The checkbox 2 is    /
        ValueType     = Boolean               / available within the /
      SetView:                                / Groupbox             /
        CheckBox (
          Text          = "Concertos")
/*************************************************************************/
    <ELEMENT> "CB_3"                          / Copied and then      /
      General:                                / modified             /
        VarName       = Var_CB_3              / The checkbox 3 is    /
        ValueType     = Boolean               / available within the /
      SetView:                                / Groupbox             /
        CheckBox (
          Text          = "Quartetts")
```

Figure 19 (Part 1 of 2). Textual Representation of the Composer - Private Data Window

```
/*********************************************************************/
   <ESET>
   <SELECT> "S_FKA_1"
     General:
       VarName       = Var_S_FKA_1
       ValueLength   = 5
     <ALT> "FKA_1"
       General:
         Value         = "FKA_1"
       SetView:
         PushButton (
           Text          = "OK")
/*********************************************************************/
     <ALT> "FKA_2"                              / Copied and then      /
       General:                                 / modified             /
         Value         = "FKA_2"
       SetView:
         PushButton (
           ActionType    = Cancel              /Note that ActionType /
           Expand        = Horizontal          /Cancel must be inserted/
           Text          = "Cancel")
     <ALT> "FKA_3"                              / Copied and then      /
       General:                                 / modified             /
         Value         = "FKA_3"
       SetView:
         PushButton (
           ActionType    = Cancel              / Note that ActionType  /
           Expand        = Horizontal          / Cancel must be inserted/
           Text          = "Exit")
/*********************************************************************/
   <ESELECT>
 <ESET>
```

Figure 19 (Part 2 of 2). Textual Representation of the Composer - Private Data Window

Build *PWS Window* SMPDATAP.

1. Click with mouse button 2 on the *PWS Window (Text)* SMPDATAP
2. Select in the context menu ***Build PWS Window***
3. Result: The *PWS Window* SMPDATAP is updated.

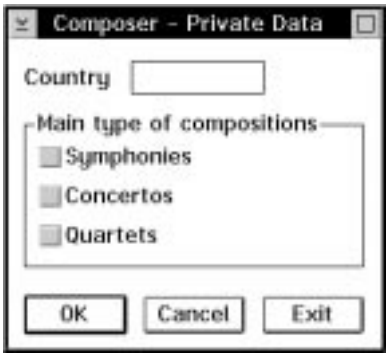
Hint

VisualLift provides syntax and semantic checks when transforming a PWS Window(Text) to a PWS Window.
When saving a PWS Window in the graphic editor a semantic check is also performed.
Only one error is reported at a time. Correcting the error and repeating the save action or build PWS Window can detect more errors.

The *PWS Window (Text)* and the *PWS Window* SMPDATAP have now identical contents.

View the modified *PWS Window* SMPDATAP.

1. Click with mouse button 2 on the *PWS Window* SMPDATAP
2. Select in the context menu ***Play***



3. Result: The *PWS Window* SMPDATAP is displayed

The *PWS Window (Text)* and *PWS Window* SMPDATAP which were updated in textual mode are now available with identical contents in textual and graphical mode.

Performing Mapping - Task 4

Up to now you have defined the attributes of a PWS Window. Now you have to *map* the elements of the PWS Window with the fields of the host screen. Mapping ensures that the information received and sent to and from the host application is handled by the corresponding workstation control.

Open the *graphic editor*.

1. Double-click with mouse button 1 on the *PWS Window* SMPDATAP object
   Result: The *graphic editor* appears

Perform mapping between Host and PWS Window.

1. Go to the SMPDATAP host screen
2. Double-click with mouse button 1 on the background of the *Composer - Private Data* window within the graphic editor
3. Result: The *Window - Attributes* notebook appears. Click-on the tab *Mapping*
4. Select host screen field at position 1842. Whenever a text appears in position 1842 on the host, this text is displayed in a message box on the workstation.
5. Click with mouse button 2 on field 1842
6. Select *Set text mapping* on the context menu. The field changes its selection state, becomes blue and is thus marked as mappable
7. Drag the field 1842 and drop it over the blue mapping area in the *Window - Attributes* notebook

   1842 appears in the *Message field position*
   Alternative: directly type in the position

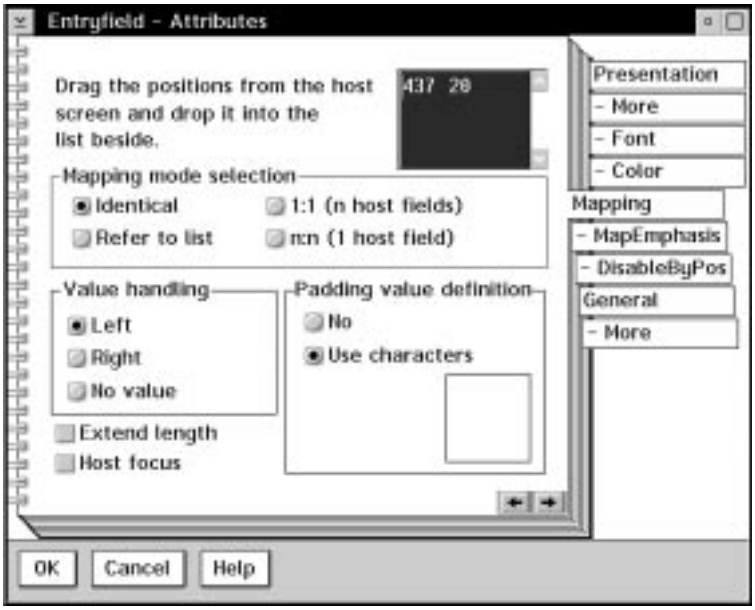
   Whenever a value appears in the host field, this value will appear in a separate message box
8. Click-on the tab *MapKeySeq*.
9. Select the ***PF9*** from the Function Key list box.

   This function key is to be sent to the host if the PWS Window is closed using the system menu
10. Select the ***OK*** push button
11. Select host screen field at position 1842

12. Click with mouse button 2 on field 1842
13. Select *Reset text mapping* on the context menu to unmark this protected field as mappable
14. Result: The Host-PWS mapping for *Composer - Private Data* window itself is complete.
15. Click on the *S* at the left of the window title to save the window.

Map host input field to PWS entry field.

1. Double-click with mouse button 1 on the country entry field of the *Composer - Private Data* window within the graphic editor
2. Result: The *Entryfield - Attributes* notebook appears. Click-on the tab *Mapping*
3. Select the input field for country at position 437
4. Click with mouse button 2 on the country input field
5. Drag the country input field and drop it over the blue mapping area in the *Entryfield - Attributes* notebook



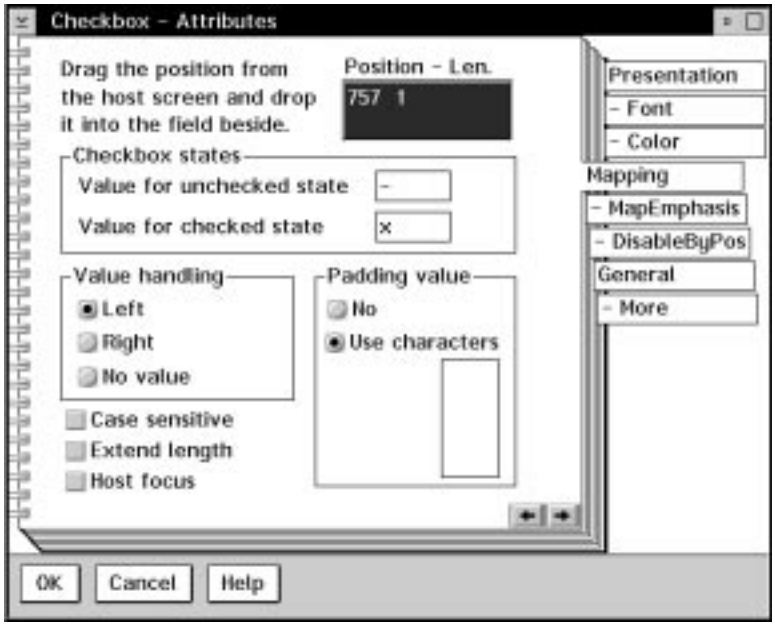
   Position and length of the host field will appear in the *Position - Length* list
   Alternative: directly type in the position and length
6. Go to *General* and change ***Max. valuelength*** to 20 as the host field is 20 characters long
7. Select the ***OK*** push button
8. Result: The Host-PWS mapping for country entry field is complete.
9. Click on the *S* at the left of the window title to save the window.

Hint

To find out which controls are already mapped use the Show Host-PWS mapping status action within the graphic editor. The controls already mapped are highlighted. This action is especially useful when a window contains more than ten controls or when interrupting the Perform Mapping task.

Map host input fields to PWS check boxes.

1. Double-click with mouse button 1 on the symphonies check box of the *Composer - Private Data* window within the graphic editor
2. Result: The *Checkbox - Attributes* notebook appears. Click-on the tab *Mapping*
3. Select on the host screen the input field for symphonies at position 757
4. Click with mouse button 2 on the symphonies input field
5. Drag the symphonies input field and drop it over the blue mapping area in the *Checkbox - Attributes* notebook

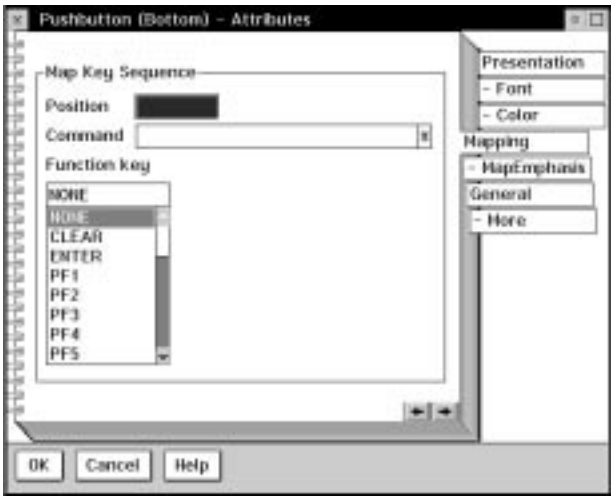


   Position and length of the host field will appear in the *Position - Length* list
   Alternative: directly type in the position and length
6. Enter an _ in *Value for unchecked state*
7. Enter an x in *Value for checked state*
8. Select the ***OK*** push button
9. Result: The Host-PWS mapping for symphonies check box is complete.

Repeat this step for the concertos and quartets check boxes using the input fields at positions 827 and 917. Click on the *S* at the left of the window title to save the window.

Map host function keys to PWS push buttons.

1. Double-click with mouse button 1 on the OK push button of the *Composer - Private Data* window within the graphic editor
2. Result: The *Pushbutton (Bottom) - Attributes* notebook appears
3. Click-on the tab *Mapping*
4. Select ***Enter*** from the *Function Key* list box



5. Select the ***OK*** push button
6. Result: The Host-PWS mapping for the OK push button is complete

Repeat this step for the Cancel and Exit push buttons selecting ***PF3*** for Cancel and ***PF9*** for Exit from the *Function Key* list box.

File the modifications of the window within the *graphic editor*.

1. Select the menu item ***PWS Window***
2. Select the pull down ***File***
3. Minimize the graphic editor
4. Result: The modified *PWS Window* (SPMDATAP) is saved to disk.

Rebuild the PWS Window (Text) SMPDATAP.

1. Click with mouse button 2 on the PWS Window SMPDATAP
2. Select in the context menu **Build Text**.

Important

Keep in mind to always synchronize PWS Window and PWS Window (Text).

View the *PWS Window (Text)* SMPDATAP and search for the **3270Map** sections. These sections contain the mapping information and have been created within this task. Like in “Designing a PWS Window - Task 3” on page 40 you could perform the mapping task in text mode, too.

In this case you can get the mapping information (e.g. position and length of controls) from the status line at the bottom of the scanned host screen.

Hint

Usually the more experienced user can combine Task 3 and Task 4.

Building a Panel ID - Task 5

For the PWS Window SMPDATAP a Panel ID has to be assigned. This Panel ID assigns the PWS Window to the host screen at run-time.

Hint

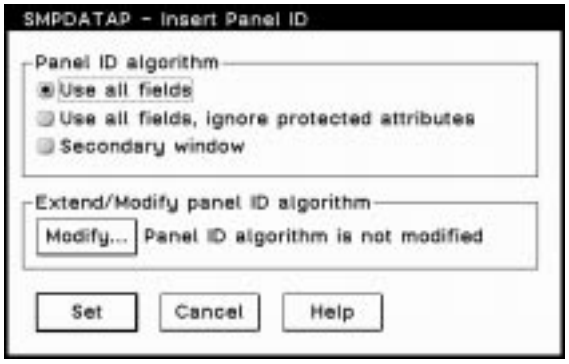
In some cases it is necessary to extend or modify the Panel ID algorithm that is provided with VisualLift. Examples for such cases are:

- The length of a host field changes dynamically at run-time
- Host fields are created or removed at run-time
- Two different host screens have an identical structure with respect to number of fields, field positions, and field length

See “Panel Identification Process” on page 14 for detailed information.

Build panel ID.

1. Click with mouse button 2 on *PWS Window* SMPDATAP
2. Select in the context menu ***Insert panel ID***



3. Result: The *SMPDATAP - Insert Panel ID* dialog box appears
4. ***Use all fields*** is preselected
5. Press the ***Set*** push button in the *SMPDATAP - Insert Panel ID* dialog box
6. Result: The *PWS Window* icon is updated
7. The PWS Window icon is visualized in two modes:
   PWS Window without panel ID.
   PWS Window with panel ID.

The PWS Window SMPDATAP is now ready to be included into the application.

Update *PWS Window (Text)* SMPDATAP

1. Click with mouse button 2 on the *PWS Window* SMPDATAP
2. Select in the context menu ***Build Text***
3. Result: The *PWS Window (Text)* SMPDATAP is updated.

Be aware

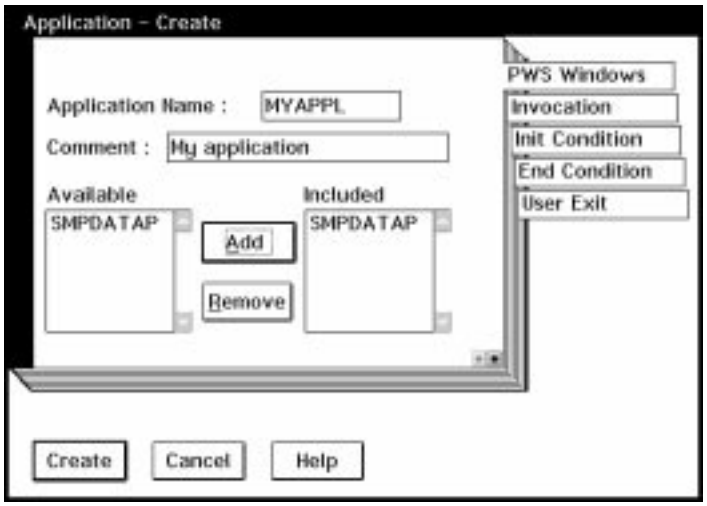
View the PWS Window (Text) object SMPDATAP. The Panel ID is visible within the PWS Window (Text). Whenever you build or modify the panel ID, build the PWS Window (Text) object. Otherwise the panel ID will be lost if you modify the PWS Window (Text) object, and build the PWS Window object from that changed textual format.

Building an Application - Task 6

You finished successfully the various tasks necessary to Visual*Lift* windows. Now the created window SMPDATAP will be incorporated into an application called MYAPPL.

Create an application object using the context menu and templates.

1. Click with mouse button 2 on the background of the *COMPOSER - Contents* window
2. Select in the context menu ***Create object***
3. Select in the cascaded menu ***Application***



4. Result: The *Application - Create* notebook appears

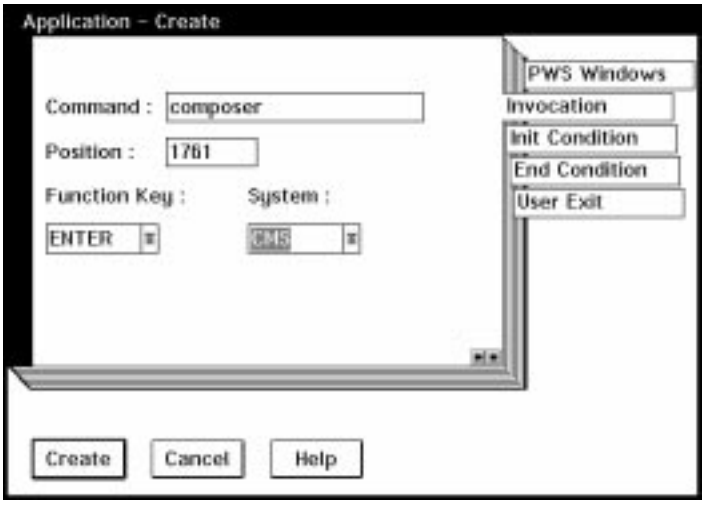
Perform the following steps in the *PWS Windows* section of the *notebook*.

1. Enter the ***Application Name*** (MYAPPL)
2. Enter (optional) a ***Comment*** (My application)
3. Select window SMPDATAP in the *Available* list
   If SMPDATAP is missing in the *Available* list check whether you assigned the Panel ID.
4. Press the ***Add*** push button
5. Result: SMPDATAP shows up in the *Included* list

Hint

If a PWS window object is missing in the Available list, you forgot to assign a Panel ID for the PWS window object. Also only these windows are shown which are part of the worked on project.

Click-on the tab ***Invocation***. Perform the following steps in the *Invocation* section of the notebook.



1. Enter the ***Command*** (composer) which starts the host application.
2. Enter the ***Position*** (1761) of the command on the host screen.

   Hint

   In order to define application invocation, init condition, and end condition, you need to know field positions in the host screens from which the application starts and on which it ends. Scan the host screens as described in Task 2 to determine these positions.
   The positions used here can be found in host screen object SMPVM in the SAMPLES project.

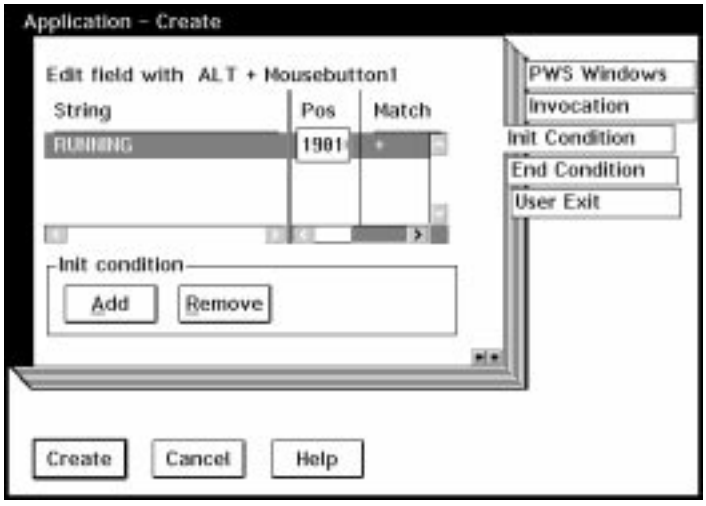
3. Select the ***Function Key*** (Enter) from the *Function Key* drop-down list. This function key will be entered when the application is started.
4. Select the ***System*** (CMS) where the host application resides from the *System* drop-down list

   Note: You can invoke an application without passing a command or a function key. When defining an application, it is allowed to specify *None* for the function key and to omit the command.

   When the application is invoked, the RTE neither issues a command nor passes a function key to the host, but just starts processing by trying to identify a host screen. This allows to start an application when one of the screens of that application is currently displayed at the host.
5. Click-on the tab ***Init Condition***.

The entries in the *Init condition* section are optional. The init conditions specify a set of strings that should be checked on the current host screen *before* the application is started. This is used to ensure that the host is in the expected state and is able to start the application. All init conditions specified must be fulfilled before the host application is started. Otherwise the invocation of the Visual*Lifted* application will be rejected.

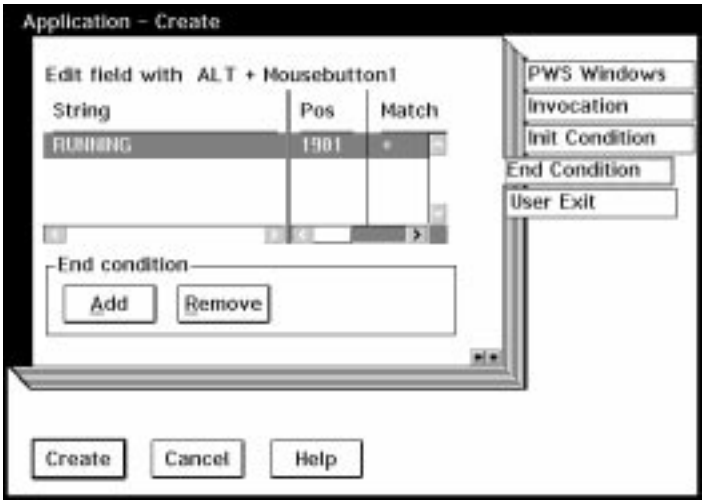
Perform the following steps in the *Init Condition* section of the *notebook*.



1. Press the ***Add*** push button
2. Press the Alt-key and click with mouse button 1 on the *String* field
3. Enter the ***String*** (RUNNING) to be checked before the Visual*Lifted* application is started
4. Press the Alt-key and click with mouse button 1 on the *Pos* field
5. Enter the ***Position*** (1901) of the string on the host screen
6. Press the Alt-key and click with mouse button 1 on the *Match* field
7. Enter a ***+***
   + means that the Init Condition is fulfilled if the specified string is found at the host position.
   - means that the Init Condition is fulfilled if the specified string is *not* found at the host position
8. Press the Alt-key and click with mouse button 1 on the *Case* field
9. Enter a ***+***
   + means that the string is checked for case sensitivity
   - means that the string is not checked for case sensitivity

Click-on the tab ***End Condition***.

The End Conditions specify a set of strings that should be checked on the current host screen to identify the termination of the application. If a host screen could not be assigned to a PWS Window running the Visual*Lifted* application, it is checked for the End Conditions. If all End Conditions that are defined in this section are fulfilled, then the Visual*Lifted* application ends. At least one End Condition should be specified. Perform the following steps in the *End Condition* section of the *notebook*.



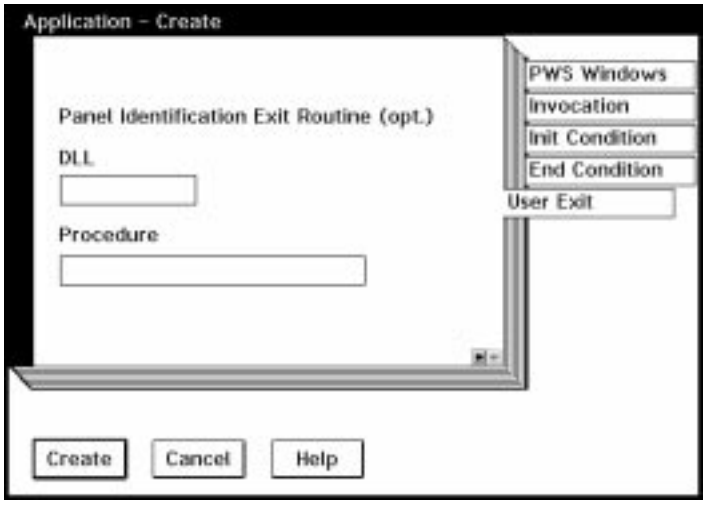
1. Press the ***Add*** push button
2. Press the Alt-key and click with mouse button 1 on the *String* field
3. Enter the ***String*** (RUNNING) of the host screen when the Visual*Lifted* application returns control to the host system
4. Press the Alt-key and click with mouse button 1 on the *Pos* field
5. Enter the ***Position*** (1901) of the string on the host screen
6. Press the Alt-key and click with mouse button 1 on the *Match* field
7. Enter a ***+***
   + means that the End Condition is fulfilled if the specified string is found at the host position.
   - means that the End condition is fulfilled if the specified string is *not* found at the host position.
8. Press the Alt-key and click with mouse button 1 on the *Case* field
9. Enter a ***+***
   + means that the string is checked for case sensitivity
   - means that the string is not checked for case sensitivity

See “Application Invocation” on page 12 and “Run-time Processing Sequence” on page 12 for additional background information.

Click-on the tab ***User Exit***.

The settings in this section define the panel identification exit routine of the Visual*Lifted* application. The specification of a panel identification exit routine is optional. For more information refer to “Supplying Application Provided Routines” on page 67.

Leave the *DLL* and *Procedure* entry fields empty.

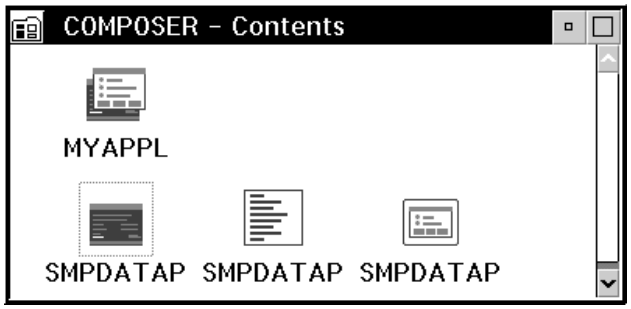


Press the ***Create*** push button

Result: The *VisualLifted Application* MYAPPL is created in the *COMPOSER - Contents* folder.

The application is bundled - and ready for test.

The application is now Visual*Lifted*. Within your *COMPOSER* project you will find the application *MYAPPL* icon: it represents the entire Visual*Lifted* application.



Testing a VisualLifted Application - Task 7

You have successfully created your Visual*Lifted* application. Now it is time for testing.

Prepare for Play.

1. Click with mouse button 2 on the application object within your project
2. Select in the context menu *Play* to initiate the *Play* sequence for all windows in your application
3. Use push buttons or menu selections to switch to the next window or to cancel the test: if you select a push-button or menu selection with an `actiontype` of *Cancel*, the play option ends. Otherwise the next window will be displayed.

Now it is time to test the Visual*Lifted* application *MYAPPL* with the *Simulated 3270 session*.

Prerequisite: Check whether in *COMPOSER - Settings* both check boxes for the 3270 EMULATOR are selected.

1. Make sure the *Simulated 3270 Session* has been started and that the initial screen is displayed (the one showing 'RUNNING BOEVM3' at the bottom)
2. Click with mouse button 2 on the application object MYAPPL within your project *COMPOSER*.
3. Select in the context menu *Test*
4. The *MAIN SELECTION* host panel appears with the text **VisualLift-A:Screen not recognized(myappl)** in the emulator window title
5. Enter **3** for *desired option*
6. The *LIST COMPOSERS* host panel appears with the text **VisualLift-A:Screen not recognized(myappl)** in the emulator window title
7. Enter **2** for *OPTION*
8. The Visual*Lifted Composer - Private data* workstation window is displayed together with the *PRIVATE COMPOSER DATA* host panel

End the test by pressing the *Exit* push button to exit the Visual*Lifted* application.

During the test mode, all host screens that could not be assigned to a PWS window or event have been captured and temporarily saved. After testing of the Visual*Lifted* application has ended and host screens could not be recognized, a dialog box appears listing these host screen captures. For more information refer to “Host Screen not recognized at run-time” on page 89.

If as a result of the test you have to modify one of your definitions, go back to either the *PWS Window* or *PWS Window (Text)* and perform the change. Then do the following:

- File the *PWS Window* (if you worked in graphical mode) and rebuild the *PWS Window (Text)*
- Build the *PWS Window* (if you worked in textual mode)
- Click with mouse button 2 on the Visual*Lifted* application and select *Rebuild* in the context menu
  Result: Your modifications are now incorporated into the Visual*Lifted* application
- Rerun the test.

Distributing a VisualLifted Application - Task 8

The host application is Visual*Lifted* and tested. Within your working environment you can use the Visual*Lifted* application. Now you want to provide the Visual*Lifted* application to other users. In general there are two things you have to do:

1. Install the run-time environment (RTE) of VisualLift on each user's workstation
2. Install the Visual*Lifted* application to the workstation.

   If you are working in a LAN environment, RTE may be installed on a LAN server, and the Visual*Lifted* application may also reside on the server and need not be distributed to each user's workstation. See “VisualLift and LAN” on page 19.

Installing the Run-Time Environment

Refer to Chapter 4, “Installation” on page 23 for detailed information on how to install VisualLift.

Installing a VisualLifted Application

There are two ways to provide the Visual*Lifted* application to other users:

- Via diskette or LAN
- Via host.

Basically a Visual*Lifted* application consists of the application file and the message file that contains the separated text strings. Additional files can be associated with a Visual*Lifted* application. For more information on these files refer to “Files of a VisualLifted Application” on page 99.

Use **Export** to copy all files belonging to an application from the workbench to another drive or directory.

1. Click with mouse button 2 on your application object.
2. Select *Export* in the context menu.

3. Result: The *Application - Export* window appears.
4. Specify the required information and press the ***Export*** push button
5. Result: The application is exported to the specified target directory

When exporting an application, two command files are automatically generated which allow installation of the exported application to a target workstation. The extension for OS/2 is .CMD, for Windows .BAT. For example, the command files for the ***MYAPPL*** application are MYAPPL.CMD and MYAPPL.BAT.

Distribution via Diskette or LAN

Run the command file to install a Visual*Lifted* application on a particular workstation

1. Switch to the directory to which the application was exported. Issue, for example, **A:** if it was exported to diskette, or, for example, **X:** and **cd APPLS** if it was exported to the APPLS directory on the LAN server drive X.
2. Install the application by issuing, for example, **MYAPPL**
3. Result: The files are copied to the directories where they must reside to be accessible by the RTE

Distribution via Host

After exporting the application, a host distribution could be performed this way:

1. Pack all exported files
2. Load the pack file to the host
3. Download the pack file to the user's workstation
4. Unpack
5. Go to the directory where the unpacked files reside
6. Issue the install command, for example, **MYAPPL**
7. Result: The files are copied to the directories where they must reside to be accessible by the RTE

Running a VisualLifted Application - Task 9

To run a Visual*Lifted* application a program object has to be created and a host session has to be assigned.

This step describes how to run a Visual*Lifted* application in an OS/2 environment. To run a Visual*Lifted* application in the Windows environment is described at “Start a VisualLifted Application” on page 73.

Creating a Program Object

Create program object.

1. Double-click with mouse button 1 on the ***OS/2 System*** icon on the *desktop*
2. Double-click with mouse button 1 on the ***Templates*** folder on the *desktop*
3. Drag the ***Program*** template object and drop it over the *Desktop*
   Result: The *Program - Settings* notebook appears

Define the settings of the Visual*Lifted* application

- For ***Path and file name*** enter the following:
  fclrts \RTS\BIN\FCLFUNC.EXE, where *fclrts* is the VisualLift root directory
- For ***Parameters*** enter the following:

  ```
  applname x [Options]
  ```

  where `applname` is the name of the application, e.g. MYAPPL,
  where `x` is the session ID of the host session. Default is A.

  See Appendix C, “Options for Running a VisualLifted Application” on page 103 for the options.
- Specify ***/FCL_EMULATOR=EMUTEST*** to make ***MYAPPL*** run against the simulated 3270 session
- Click-on the tab ***General***.
- Enter in the ***Title*** entry field the name of the object representing the Visual*Lifted* application
- Modify parameters in other sections of the notebook according to your needs
- Double-click with mouse button 1 on the ***System Menu*** of the *Programs - Settings* notebook
  Result: The Visual*Lifted* application icon is on the desktop

Move the Visual*Lifted* application icon to the desired area of the desktop

Double-click with mouse button 1 on the icon to start the Visual*Lifted* application.

Congratulations

You successfully ran your host application with the new user interface you designed.

Optional Tasks

The previously described tasks are required to Visual*Lift* a host application. The optional tasks can enhance your Visual*Lifted* application.

Defining Events

Events allow you to describe conditions that are not covered by any data in the description file of the currently running application. Examples for events can be:

- Messages from the operator that make the current screen disappear
- The currently running Visual*Lifted* application is left (temporarily or permanently) and another Visual*Lifted* application is entered. The event file of the currently running application is closed, and the event file of the target application is opened.
- Messages must be translated, for example, to replace host specific terms with workstation terms

There are two types of event files: the system event file and one event file per application. To maintain system and application event files follow these steps:

- Click with mouse button 2 on the background of the VisualLift Workbench folder (for the system event file)
- Click with mouse button 2 on the application object within your project folder (for the application event file)
- Select the action *Open event file* from the context menu.

This action opens the text editor that is defined in the VisualLift Workbench Settings. You now can edit the event file.

There are application event files for both sample applications in the SAMPLES project.

Refer to the VisualLift Reference and the VisualLift Solution Guide to learn more about the purpose of events, event handling, and the event definition syntax.

Supplying Application Provided Routines

VisualLift provides a variety of exits. You can supply your own routines (application provided routines) using these exits. The different types of application provided routines are:

- Panel ID exit routines to perform your private host panel identification, event processing, and application end processing
- Mapping routines to perform your private mapping between host and workstation values
- Checking routines to validate input values on the workstation without involving the host application
- Action routines to integrate and communicate with other workstation based applications
- Help exit routines to provide special help handling facilities
- Window procedures to integrate non-standard workstation controls, for example business graphics.

Application provided routines have to be written in the C language. The code must reside in a dynamic link library (DLL). The routines are made known to VisualLift by specifying the DLL and procedure name of each exit.

The file SAMPLE.C in directory *fclroot*\FCL\USEREXIT contains samples for all types of application provided routines.

The following objects of the SAMPLES project show how the names of DLL and PROCEDURE are defined.

- SMPCOMPO for an action routine and a mapping routine
- SMPADDCO for a checking routine
- SMPMAIN for a help exit routine
- SMPSTATS for a window procedure
- SMPAPPL and SMPPRTAP applications for panel ID exit routine

Panel ID exit routines are defined on the last page (***User exit*** tab) in the Settings notebook of an application object. See page 14 for more information about the panel identification process.

Refer also to the VisualLift Reference to learn more about the different application provided routines. There is also information available about the VisualLift provided services that can be used within application provided routines. Use these services to access and maintain values in host fields and workstation controls.

Supplying Help

You can provide the following types of help for your Visual*Lifted* application:

- *General help*, *Keys help*, and *Using help* for each PWS Window
- *Context help* for each control within a PWS Window.

A general help, keys help, or using help is displayed by the RTE whenever a push button or menu choice in the window is selected which is defined with *ActionType = Help*, *ActionType = KeysHelp* or *ActionType = UsingHelp*. The context help for the control that currently has the input focus is displayed by the RTE whenever the F1 key is pressed. The OS/2 Information Presentation Facility (IPF) is called by the RTE to manage the help display. In Windows, the appropriate Windows functions are called. This enables you to use all the features, for example hypertext-links that are available with the respective help manager. To enable help do the following:

1. Specify the name of the help file containing all help texts within your window definition. It is recommended to use the same help file for all windows of one application.
2. Specify the numeric help ID for each help within your window definition. For each help ID there must be the corresponding ID in the help file identifying the help window that is displayed by the RTE.
3. Write the help source file:
   a. For OS/2 the extension of help source files is IPF
      Use the IPF Compiler (IPFC) to generate a help file (extension HLP) from the source file. The IPF Compiler is part of the IBM OS/2 Developer's Toolkit.
   b. For Windows the extension of help source files is RTF
      Use the Help Compiler (HC) to generate a help file (extension HLP) from the source file. The Help Compiler is part of the Software Development Kit (SDK) for Windows.
4. The help files must be stored in the language directory:
   fclroot\FCL\NLSDATA\ENGLISH*.HLP

The file SAMPLE.IPF in directory *fclroot*\FCL\USEREXIT contains a sample help source file.

The object SMPMAIN of the SAMPLES project shows how help is defined for a PWS Window.

If you want to provide your application in multiple languages translate and process your help file equivalent to the tasks that are described in section “Supporting Multiple Languages.”

Supporting Multiple Languages

VisualLift enables you to translate the workstation interface into different languages. This process is independent of the language of the host application interface and can be performed without modifying the host application.

All text strings specified within a window definition are extracted by the VisualLift ADE and stored into an NLS file. The separated text strings are subject of translation into other languages. NLS files have the same name as the PWS Window object and are stored in the NLS directory (with the extension NLS):
fclroot\FCL\NLSDATA*.NLS

The VisualLift ADE also generates a machine readable MSG file from the NLS file. MSG files have the same name as the PWS Window object and are stored in the language directory (with the extension MSG):
fclroot\FCL\NLSDATA\ENGLISH*.MSG

Once the NLS files have been translated into the new language, you must generate a new MSG file using the MKMSGF utility. MKMSGF is part of the IBM OS/2 Developer's Toolkit.

The MSG files are processed unchanged also by the Windows RTE of VisualLift.

Rules for Translating NLS Files

Adhere to the following rules when translating the NLS files:

- The first line of the file contains an identifier. Do not translate this line.
- The second line starts with an identifier for the message number. Do not translate the identifier, only the text following the identifier.
- Do not change the sequence of the strings.
- Do not delete or add lines.
- Each string must stand alone on one line.

Support Multiple Languages Concurrently

VisualLift allows to install Visual*Lifted* applications in multiple languages on one workstation. Take the case that your application is English and has to be translated into French. Now you want to install both languages on one workstation. To do so, you have to create a second language directory, for example: *fclroot*\FCL\NLSDATA\FRENCH

***.MSG** The English MSG files must reside in directory: *fclroot*\FCL\NLSDATA\ENGLISH

***.MSG** The French MSG files must reside in directory: *fclroot*\FCL\NLSDATA\FRENCH

The language used by VisualLift is determined by the environment variable `FCLLANGUAGE`. The value of the environment variable is the name of the language directory.

`SET FCLLANGUAGE=ENGLISH` To use the English version of the Visual*Lifted* application specify the environment variable in the CONFIG.SYS file.

`SET FCLLANGUAGE=FRENCH` To use the French version of the Visual*Lifted* application specify the environment variable in the CONFIG.SYS file.

If `FCLLANGUAGE` is not set, ENGLISH will be used as default.

Translating VisualLift RTE Messages

Messages and texts issued by the VisualLift RTE will appear in English. This information can be translated into another language. The file containing the messages and text strings is named `fclm90en.src` and is located in the directory *fclroot*\FCL\USEREXIT

The corresponding machine readable message file is named `fclm90en.msg` and is located in the directory *fclrts* \RTS\MSG

After translating `fclm90en.src` into the new language, you must generate the new message file using `mkmsgf` utility. `mkmsgf` is part of the IBM OS/2 Developer's Toolkit. The rules to translate the `src` file are described in “Rules for Translating NLS Files” on page 69.

The message file is processed unchanged also by the VisualLift Windows RTE.

Chapter 7. VisualLift RTE for Windows

The RTE of VisualLift is available under Windows - Visual*Lift* a host application with the VisualLift ADE (under OS/2) and run the Visual*Lifted* application either under OS/2 or Windows. The installation of VisualLift RTE for Windows is described in the installation description file FCLREADM on the host. For the system requirements of a workstation using VisualLift RTE for Windows refer to Appendix A, “System Requirements” on page 97.

Basically, the functions of the RTE of OS/2 and Windows are identical. But the different approaches of the operating systems and user interface technology cause deviations. The deviations to the OS/2 version of VisualLift RTE and special features you need to be aware of are described in this chapter.

Deviations of VisualLift RTE for Windows

VisualLift RTE for Windows has deviations compared to VisualLift RTE for OS/2. VisualLift RTE for Windows:

- Allows only one Visual*Lifted* application to run at one time
- Does not support double-byte character set
- Allows no scrolling and re-sizing of Visual*Lifted* windows.

Windows Considerations for VisualLift ADE

The following items have to be considered when you Visual*Lift* a host application that is to be used in the VisualLift Windows RTE.

User Interface

The following user interface related items have to be considered when you Visual*Lift* a host application that is to be used in the VisualLift Windows RTE.

- Valueset and notebook cannot contain icon files
- Spin button and push buttons cannot be borderless
- Graphic controls are available for bitmaps (icon and metafile files are not supported)
- Some restrictions for fonts and colors

The Windows considerations for each specific control are mentioned at the corresponding control in Online Solution Guide as well as in the VisualLift Reference.

Application Provided Routines

Application provided routines to be used by the Windows RTE require the use of the Windows DLL format. The *Software Development Kit (SDK)* for Windows provides tools to generate that format. VisualLift provides the members FCLXUSER.LIB, FCLXWIN.H, and FCLXUSER.H in the directory *fclroot* \FCL\USEREXIT

This directory also contains four files with the file name FCLWSAMP (extension C, MAK, DEF, LNK) which contain an example how to build a Windows DLL. As these

DLLs are produced by Windows, the *Export* function does not cover them. Also .HLP and .WCO files are not covered by the *Export* function.

Help Information

The help information for a Visual*Lifted* application has to be created in the Windows format. A word processor supporting the *Rich Text Format (RTF)* and tools of the SDK for Windows are used to provide help information in Windows format.

Help for a Visual*Lifted* application is requested by selecting the **Help** push button; context-sensitive help (invoked by pressing **F1**) is not supported.

Environment Variables

The VisualLift Windows RTE does not require that you set environment variables or search paths in the `AUTOEXEC.BAT` file. Although it is not required, you can set all environment variables of OS/2 (specified in `CONFIG.SYS`) in the `AUTOEXEC.BAT` file.

Rules for Setting Environment Variables

If `FCLROOT=` is not set, VisualLift defaults to the directory which is used as *working directory* in the program item.
If `FCLRTS=` is not set, VisualLift defaults to the directory which is used as *working directory* in the program item.

Increase Environment Space

If you set environment variables and the environment space is too small, you can extend it by the following statement in the `CONFIG.SYS` file:

```
SHELL=C:\DOS\COMMAND.COM /e:512 /p
```

`/e` indicates the size of the environment space. Increase the number to accommodate all `SET` statements.

Minimize Icons

VisualLift Windows RTE supports the minimizing of Visual*Lifted* windows to icons. Consider the following:

- The minimized icon is specified by the parameter `icon=`
- The icon formats of OS/2 and Windows are not identical. The Windows icon must be created with an appropriate Windows tool, for example the image editor (IMAGEDIT.EXE) of the SDK. If you use WINOS2, try to copy the OS/2 icon into the clipboard and paste the icon in the Windows environment from the clipboard to a file.
- The Windows icon must have 32 x 32 pixels containing 16 colors. A good check for correctness is the file length: 766 bytes.
- To avoid confusion between OS/2 and Windows icons the Windows icons must have the extension WCO (instead of ICO). In the textual representation, however, the extension is ICO in order to allow identical application files.

Graphic File Formats and Location

In general you can use your OS/2 2.0 (and higher) bitmap format unchanged because the Windows 3.1 bitmap format is compatible.

There are two ways to check whether the bitmap file format is correct:

- Try to load the bitmap file into PAINTBRUSH (PBRUSH.EXE part of Windows) or appropriate Windows tool
- Try to use the bitmap file as screen background.

The bitmap files (extension BMP) and the minimize icon files (extension WCO) must be located in the directory *fclrts* \RTS\NLSDATA\ENGLISH

Space and Time Performance

The VisualLift RTE for Windows can use decompression routines to:

- Save disk space
- Speed up the performance

when using a Visual*Lifted* application.

Each Visual*Lifted* application is a file in the format *filename*.AST. Compressing this file generates a file of the format *filename*.AS_. To take advantage of the compression, you have to use the COMPRESS utility (part of the Windows SDK) to compress each Visual*Lifted* application file.

The compressed Visual*Lifted* application file requires only about 30 % of the disk space of an un-compressed file. On fast processors, it is recommended to use the compressed format to save disk space and to improve performance.

VisualLift automatically distinguishes compressed and un-compressed formats. If both formats (*filename*.AST and *filename*.AS_) are available, VisualLift gives priority to the compressed format.

Start a VisualLifted Application

Before using a Visual*Lifted* application within the VisualLift Windows RTE, you have to create a program item. The prerequisites are:

- That the VisualLift Windows RTE is installed on your workstation
- That the Visual*Lifted* application files are installed as described in Appendix B, “System Information” on page 99.

You have to perform *Create a program item* for each Visual*Lifted* application. The result is that the Visual*Lifted* application icon is located in the **VisualLift** group. Double-clicking on the program icon starts the Visual*Lifted* application.

Create the Program Item

1. Open the ***VisualLift*** group window created during the installation
2. From the ***File*** menu, select ***New***
3. In the ***New Program Object*** dialog box, select the ***Program Item*** option, and then select the ***OK*** button.

4. Fill in the ***Program Item Properties*** dialog box as necessary, and select the ***OK*** button.

For ***Command Line*** enter the following:

`VLW [ application [ session ] ] [ parameters ]`
where `VLW` is the name of VisualLift Windows RTE program (VLW.EXE).
where `application` is the name of the Visual*Lifted* application, e.g. APPL. If the application name is omitted, you will be prompted to specify it.
`session` is the session ID of the host session. Default is A.

See Appendix C, “Options for Running a VisualLifted Application” on page 103 for the parameters.

For ***Working Directory*** enter the following:

D:\FCL is the directory where the VisualLift Windows RTE resides.

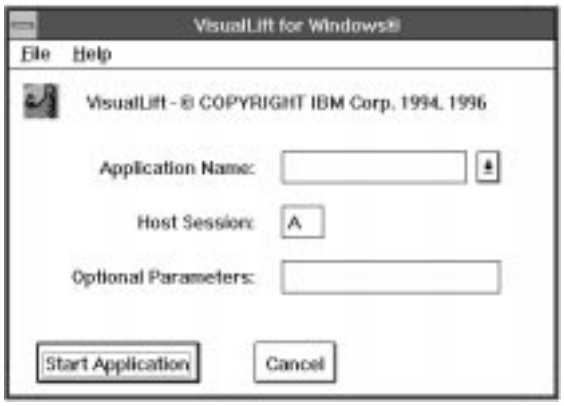
Hint

Specifying the Working Directory eliminates the need to set the path (in AUTOEXEC.BAT) and the environment variables.

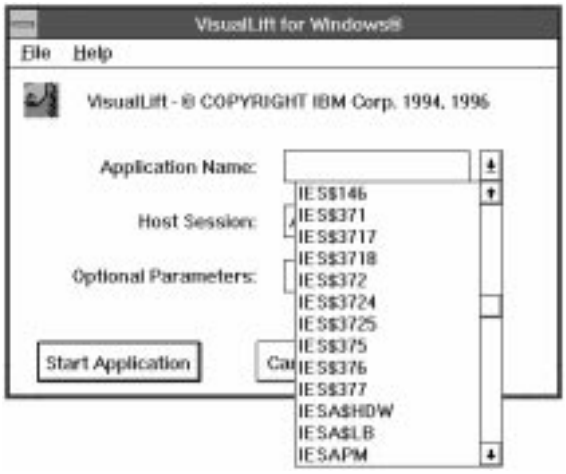
Start Applications without Specification

Starting a Visual*Lifted* application can be done by associating the application name, the session, and parameters with the VisualLift Windows RTE as described in “Create the Program Item” on page 74.

Another way to start applications is to create a program item as described in “Create the Program Item” on page 74 and not specifying the *application*, *session*, and the *parameters*. Double-click on the VisualLift icon and the following window appears:



Specify the *Application Name*, the *Host Session*, and *Optional Parameters* to start an application. The application name can be selected from the drop-down combination box as shown in the following window. (The names shown in the drop-down combination box below are examples).



Pre-loading VisualLift Libraries during Startup

The startup time for the invocation of VisualLift can be reduced by loading the resident parts of VisualLift during Windows startup. Perform one of the following choices to pre-load VisualLift:

1. Create a program item in the ***Startup*** group.
   a. Open the ***Startup*** group window.
   b. From the ***File*** menu, select ***New***
   c. In the ***New Program Object*** dialog box, select the ***Program Item*** option, and then select the ***OK*** button.

   d. Fill in the ***Program Item Properties*** dialog box as necessary, and select the ***OK*** button.

   Command Line

   `VLWHOLD` is the name of the program.

   Working Directory

   `D:\FCL` is the directory where VLWHOLD resides.

   e. The VLWHOLD object is displayed in the ***Startup*** group window.

2. Edit the `win.ini` file. In line `run=` enter *fclroot*\VLWHOLD.
   For example: `run=d:\fcl\vlwhold`

How to Free Space Occupied by VisualLift

For a fast startup VisualLift loads its code libraries only once and keeps them in Windows virtual memory available for the next start. Therefore repeated starts run very fast.

If you need space for other applications, you can free the space by closing the icon showing VisualLift with a lock (see page 76 for an example of such an icon). Next time you start VisualLift, the code libraries have to be loaded again, but in the meantime you can use the released space.

Chapter 8. VisualLift and Automated Build Process

As an alternative to using the VisualLift Workbench, some of the development tasks may be performed in batch mode. In a workstation development environment usually a library system is used to store the source code and maintain multiple versions. An automated build process can be run against that library.

There are different types of source code that make up a Visual*Lifted* application, like textual format of window definitions (extension UIT), code of application provided routines (extension C), event files (extension EVT), etc.

Some of the deliverables of a Visual*Lifted* application, like help files, icons, or DLLs for application provided routines, are generated using tools that are provided by the operating system or the respective toolkit. PWS windows and the application itself can be only generated by using functions of the VisualLift Workbench. To enable these objects to be processed during an automated build process as well, VisualLift provides a set of commands which can be issued from the OS/2 command line or from a command procedure. The following sections describe the purpose and syntax of those commands.

Generate PWS Window (Text) from PWS Window

If you do not use the textual window definition format, you can use the FCLGUIDX command to generate the textual representation and then store the text file in a library. The command syntax is:

```
FCLGUIDX drive:\directory\WINDOW.UIT
```

The target file that should contain the textual representation must be fully specified. The file name has to be the name of the PWS Window object.

Generate Textual Application Format from Application

To be able to store a Visual*Lifted* application, especially its settings, in a library, you can use the FCLGUIDX command to generate the textual representation of the application. Note that no textual representation of an application is used when working with the VisualLift Workbench. The command syntax is:

```
FCLGUIDX drive:\directory\APPLICATION.UIT
```

The target file that should contain the textual representation must be fully specified. The file name has to be the name of the Application object.

Generate PWS Window from PWS Window (Text)

To generate a Visual*Lifted* application, all windows that are part of this application have to be available. The FCLASTG command generates a PWS window from the textual representation of that window. The command syntax is:

```
FCLASTG drive:\directory\WINDOW.UIT
```

The file containing the textual representation must be fully specified. A PWS Window object having the same name as the text file will be generated.

Generate Application from Textual Application Format

The FCLAPLG command generates an application from the textual representation of that application. Note that no textual representation of an application is used when working with the VisualLift Workbench. The command syntax is:

```
FCLAPLG drive:\directory\APPLICATION.UIT
```

The file containing the textual representation must be fully specified. An application object having the same name as the text file will be generated.

Check Semantical Correctness of a PWS Window

When generating a PWS window from the textual representation within the VisualLift Workbench, the window definition is also checked for semantical correctness. The FCLASTG command does not include this check, but you can use the FCLCCX command for this purpose. The command syntax is:

```
FCLCCX window
```

Only the name of the PWS Window object must be specified.

Chapter 9. VisualLift Hints and Tips

Tricks of the trade are described within this chapter. The following hints and tips might avoid problems during your work with VisualLift. The sequence of hints is organized according to the following structure:

- General hints concerning VisualLift
- Workbench related hints
- Graphic editor related hints
- RTE related hints.

General Hints

Learning to use VisualLift effectively
The first host application to be Visual*Lifted* should be a small one (recommended size: up to 10 screens) or a subset of a large application. When the small host application is successfully Visual*Lifted*, then Visual*Lift* a larger host application.

How many controls are advisable within one window?
The average number of controls within one window should not exceed 25 controls. Significantly more controls will cause usability problems. If you have to use more than 40 controls on the host screen think about:

- Using one or more secondary windows or
- Using a notebook.

How many windows are useful within an application?
50 windows. Extremely large applications may cause performance problems. On average, an application consisting of about 50 windows will have no performance problems - provided that each window contains up to 25 controls.

How to stop the storage resident parts of VisualLift?
Enter in the OS/2 command prompt: FCLWMSSE STOP
Under Windows close the icon showing the lock (see page 76 for an example of such an icon).

VisualLift Workbench

Export and Import

The export and import functions are designed for projects and applications known to VisualLift. The following object types and corresponding files are recognized by VisualLift:

Scanned Host Screens	*.SCN
PWS Windows (Text)	*.UIT
PWS Windows	*.AST and *.MSG and *.NLS
Applications	*.AST and *.MSG
Event Files	*.EVT
Workbench Files	*.PRJ and *.INI

Select *Export referenced files* in the corresponding Export window to also export graphic files (*.BMP, *.ICO, *.MET), help files (*.HLP), and application provided routines (*.DLL).

Exclude and Include

Include and exclude means that objects are made visible or not visible. The same object can be contained in several project folders.
Workbench objects with identical names are *physically* identical - even if they are contained in different projects folders.

Transformation

You can convert the *PWS Window (Text)* object into a *PWS Window* object - or vice versa. This offers you freedom in how to use VisualLift. On the other hand it is your responsibility to synchronize source object(s) and converted object(s). Whenever you try to convert either a PWS Window or a PWS Window (Text) object to a target with a *newer* date, a confirmation window informs you that you might overwrite changes.

Recommendation

For the novice user of VisualLift the *PWS Window* is the ***source*** object. The result of conversion is a *PWS Window (Text)* object.

For the experienced user of VisualLift the *PWS Window (Text)* is the ***source*** object. The result of conversion is a *PWS Window* object.

Any update of either object should be propagated *immediately* to the corresponding object. For example, if a new Panel ID is inserted into a *PWS Window* object, the corresponding *PWS Windows (Text)* object should be built.

PWS Window (Text) with Include Statements and Comments

Comments in a PWS Window (Text) are eliminated when transforming the PWS Window (Text) to a PWS Window.

Whenever a PWS Window (Text) is transformed into a PWS Window, the <INCLUDE> statements are resolved. If an error occurs during the transformation, the corresponding messages point to a line of the PWS Window (Text) file. In the directory *fclroot*\FCL\TMP a file with identical name is created which contains all

resolved <INCLUDE> statements - in PWS Window (Text) format. Examining this file helps to locate the error.

When performing the transformation sequence PWS Window (Text) ⇒ PWS Window ⇒ PWS Window (Text) any comment or <INCLUDE> statement that has been used in the original textual representation will *not* be restored.

Designing a PWS Window - FAST!

This is a way to create a PWS window from the scanned host screen object really fast. Just click-on with mouse button 2 on your host screen object and select *Build PWS window*. This creates a PWS window which shows all input fields as entry fields. PF keys are shown as push buttons. Any other controls are not created. Also the panel ID has been already assigned.

See this as additional help for designing a PWS window - it does not replace Task 3 *Designing a PWS Window*!

Refresh

The refresh action of the VisualLift Workbench rebuilds the internal PWS Window list and the internal application list. These internal lists are used to reference PWS Windows and applications in the VisualLift Workbench for performance reasons. A refresh is required if PWS Windows or application files (extension `AST` or `MSG`) are added, removed, or renamed *without* control of the VisualLift Workbench.

Graphic Editor Hints

Drag and Drop

The *drag and drop* technique is used within VisualLift. For learning this interaction technique it is recommended to study the *OS/2 Tutorial*.

Window

Open the *Window Definition Attributes* for the window with a double-click on the background within the window. Define the window definition attributes within the dialog sequence.

Dynamic Entry Fields

Split up dynamic entry fields (larger than 80 characters) in fields with up to 80 characters or use a multi-line entry field - otherwise the window size may not fit onto your screen (depending on the monitor and the graphic card you are using).

Accelerator Keys

When Visual*Lifting* an application it is a good idea to support the same keys as the host application, e.g. PF keys. When learning the new user interface, the users are still able to use the application in their known way.

Controls and Group Controls

You can use the following controls only inside their corresponding group controls:

Control	Group Control
Radio Button	Layout(Group) or Static(GroupBox)
Layout(Column)	Layout(ColumnGroup)
Layout(ColumnHeading)	Layout(ColumnHeadingGroup)
Layout(ColumnHeadingGroup)	Layout(ColumnGroup)
NoteBookPage	NoteBook

Message Mapping

Within the graphic editor double-click with the mouse button 1 on the background of the window. This action causes the *Window - Attributes* notebook to appear. Go to the mapping section.

Select now the field in the host screen which is to be mapped as a workstation message window.

Drag the host field from the opened Host Screen object and drop it over the mapping region. The corresponding position and length values are displayed in the Mapping position/length field.

If you define the position of the message field in the Host Screen as a message in the PWS Window, the run-time environment is notified whenever a value appears in the host field. The value is treated as a message text and is displayed in a separate message box.

Deleting Controls

To delete a control out of your window, drag it to the shredder.

RTE Hints

Terminal Models

The terminal models defined in the emulator session should be identical for Visual*Lifting* a host application and running a Visual*Lifted* application. In some cases host screens are not recognized by VisualLift if the host screen sizes are different. For example, an application developed on a 32x80 terminal may not run on a 24x80 terminal.

Monitor Resolution

When developing and using the Visual*Lifted* application on workstations, use identical monitor resolution for developing and running the Visual*Lifted* application. Otherwise the window layout may be unpredictable.

- When developing a Visual*Lifted* application on a workstation with higher resolution monitors (for example, XGA or SVGA) the Visual*Lifted* application runs on workstations with higher resolution monitors. Running the same Visual*Lifted* application on workstations equipped with VGA produces windows which might be larger than the physical screen. This has to be taken into consideration especially for the Windows RTE, because there the window is neither scrollable nor sizeable.
- When developing a Visual*Lifted* application on a workstation with VGA, the Visual*Lifted* application will run on a workstation with VGA and XGA/SVGA. The appearance may vary due to different font resolutions.

Chapter 10. Problem Determination

Common Problems

The symptoms you notice when working with VisualLift are not necessarily confined on VisualLift - problems may come from, for example, incorrect CONFIG.SYS (for Windows AUTOEXEC.BAT) entries.
This chapter provides solutions for the most common problems.

VisualLift OS/2 Symptoms:

VisualLift Windows RTE Symptoms:

Trouble Log

If an error occurs in a VisualLift component, an error message is created and recorded in a file. This file contains approximately the latest 50 messages. The file is called *Trouble Log*. The purpose of the trouble log is to assist you in problem determination.

The trouble log is the central point of information within VisualLift when errors occur. The trouble log file is located in *fclroot*\FCL directory and is named FCLMSG.LOG - whenever something does not perform the way you expected it, invoke the trouble log by:

- Clicking the mouse button 2 on the background of the *VisualLift Workbench - Contents* window
- Selecting ***Inspect trouble log*** in the context menu
- The trouble log viewer is opened displaying the VisualLift error messages
- Double-clicking on a specific error message opens the explanation window.

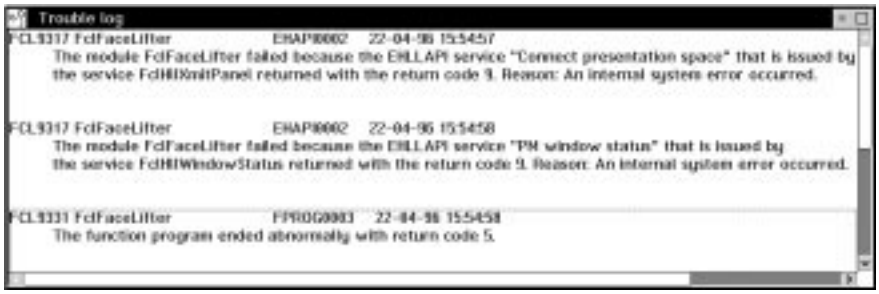


Figure 20. The Trouble Log File

Trouble Log within OS/2 RTE

Within the RTE the error messages are displayed via the trouble log viewer, too. However, you cannot invoke the trouble log viewer explicitly. The trouble log is a file named FCLMSG.LOG, is an ASCII file, and is located in the directory *fclroot*\FCL. Whenever an error occurs you get the choice to open the trouble log. You can also request the message explanation with a double-click on a specific error message. You can also view the contained error messages with any text editor.

Trouble Log within Windows RTE

Like in OS/2 the trouble log is a file named FCLMSG.LOG, is an ASCII file, and is located in the directory *fclroot*\FCL.

You can view the contained error messages with any text editor or with the NOTEPAD.EXE (part of Windows). You can delete messages from the trouble log, or you can delete the trouble log itself. Note that the latest messages are located at the bottom of the trouble log.

VisualLift OS/2 Symptoms

Disk full

Solution: When running several applications concurrently, OS/2 uses the SWAPPER.DAT. If a warning message appears, telling you that no disk space is available, do one or more of the following:

- Close other applications
- Move the SWAPPER.DAT to another disk (do not forget the CONFIG.SYS change and to reboot your workstation)
- Delete data on the disk drive where the SWAPPER.DAT is located.

As a rule of thumb, the SWAPPER.DAT size may increase by more than 10MB when running VisualLift.

Low performance

Solution: Low performance can be caused by various reasons. The most important ones related to VisualLift are:

- Too many controls within one window
- Too many windows within a Visual*Lifted* application

Scrambled tables within VisualLift Help or Reference

Solution: The tables within the VisualLift Reference and Help information are built using a monospaced font. If no monospaced font is installed on your system the information presentation facility uses the system proportional font instead. Install the System Monospaced font.

Host Screen not recognized at run-time

Solution:

- Is the host screen appearing at run-time identical with the host screen used when building the application?
- Is the screen recognized if the panel id algorithm *Use all fields, ignore protected attributes* instead of *Use all fields* is used for building the panel id? See “Building a Panel ID - Task 5” on page 57 how to do this. Rebuild the application then.
- Has the PWS window been included into the list of PWS window objects of the application? See “Building an Application - Task 6” on page 58 how to include a created window.
- Did you correctly specify the identifier of the host session? See page 35 if the application starts from the workbench. See page 66 for specifying the session ID of the host session if the application starts from its own program object.
- Did you develop an application on a different terminal model than the one you use for testing now? For example, you developed the application on an 32x80 terminal and test it on an 24x80 terminal. It usually causes problems if the host application dynamically exploits different terminal models.

Do the following if none of the above applies:

- Run the application in test mode (within the VisualLift Workbench).
- Navigate to the point within the application where the host screen could not be recognized.
- Close the application: a dialog appears listing all host screens not recognized during the test run.
- Select the host screen you want to analyze and press ***Save***

- Open the project folder that contains the application.
- Include the host screen object that could not be recognized.
- Open the host screen object that could not be recognized.
- Press mouse button 2 on the host screen window.
- Select ***Select*** in the context menu.
- Select ***Compare objects*** in the cascading menu.
- In the *Host Screen - Compare* dialog box select the saved host screen.
- Press the ***Compare*** push button.
- All host fields that do not match the specified Panel ID for the PWS Window get selected state emphasis - these are the differences between the two windows.
- Modify the panel identification, for example, by excluding these regions.

Using this feature allows you to identify those regions of the host screen that dynamically change at run time.

Host Screen not recognized, end of application instead

VisualLift application seems to have ended, but *Screen not recognized* appears in the 3270 emulator window.

Solution: VisualLift application is still active and treats the currently active 3270 screen as 'not recognized'.

- Check the application end conditions of the VisualLift application. See page 60 for specifying *end conditions*.

Workstation windows are too large

Solution: The monitor resolution has to be identical for Visual*Lifting* a host application and running a Visual*Lifted* application. Different monitor resolutions may result in a different window layout. Keep in mind:

- When developing a Visual*Lifted* application on a workstation with higher resolution monitors (for example, XGA or SVGA) the Visual*Lifted* application runs on workstations with higher resolution monitors. Running the same Visual*Lifted* application on workstations equipped with VGA might result in windows which are larger than the physical screen.
- When developing a Visual*Lifted* application on a workstation with VGA, the Visual*Lifted* application will run on a workstation with VGA and XGA/SVGA.

Another application is already active

Several VisualLift applications cannot use the same host session concurrently

Solution: Check the OS/2 task list and close any VisualLift application using the host session. Restart your VisualLift application.
Also check if the application end conditions are correctly defined. See page 60.

Message 'Cannot start FCLXMAP.EXE ...' or SYS1804

VisualLift cannot be started as VisualLift dynamic link libraries (DLLS) are not set in the LIBPATH of the CONFIG.SYS

Solution: Reboot your system. If this is not successful, delete VisualLift and repeat the installation to ensure that the paths are correctly set in the CONFIG.SYS. Make sure to select the option *Update CONFIG.SYS* when installing VisualLift.

Check if the environment variables are set (see “Environment Variables” on page 101).

The graphic editor abends

Solution:

If the PWS window is corrupted, rebuild the PWS Window from the PWS Window (Text)

Error when calling a newly imported application

and

Error when calling an application within a newly imported project

Solution: Perform a *Refresh Workbench* (see context menu on page 34)

VisualLift gets timeout message

Solution: Increase the value of the OS/2 ***FCL_TIMEOUT*** variable which is set in the CONFIG.SYS. Default is 30 seconds. You can also overwrite the timeout value by defining the `/FCL_TIMEOUT=timeoutval` option for the application's program object (see page 66.)

VisualLift starts and immediately ends

Solution: The following can cause this behavior

1. Check the path defined in the program object from which VisualLift started. The path may, for example, point to a directory where VisualLift was previously installed.
2. Neither an invocation key nor invocation command (see page 59) is defined for the application. Also the End Condition specifies a state which is true on the host screen when the application starts. In this case the end condition is true and VisualLift ends.
   Define the invocation command, invocation key, and specify different end conditions. Specify *CMS* when the application resides on a CMS system.
3. The first host screen update after the application invocation causes the end condition to be true, and the application ends. This may happen if the host application clears the screen before displaying the first screen. Then the empty screen meets the application end criteria.

Message file not found

Message file FCLM90EN.MSG could not be found

Solution:

1. Check whether the OS/2 variable FCLROOT has been set
2. Check whether the message file FCLM90EN.MSG is in *fclroot*\RTS\MSG, and reinstall the message file, if necessary
3. Check whether the message file is broken and reinstall it

Application not found

Solution: Check whether these two files are available:

The Visual*Lifted* application file (either, for example MYAPPL.AST) is located in the directory *fclroot*\FCL\ASDATA
The associated message file is located in the directory
fclroot\FCL\NLSDATA\ENGLISH

No connection to host emulator

Solution:

- Make sure the host emulator is started and a session is established *before* you start the Visual*Lifted* application
- When testing from the workbench:
  - Check your project settings and check if the simulated 3270 session is correctly specified.
- Check whether the correct *session ID* is specified (in most cases *session A*)
- The directory where the EHLLAP.DLL of the host emulator code resides *must* be in the current LIBPATH.
- If you have more than one host emulator or multiple versions of one emulator installed, make sure that only the correct one (but not the other ones) is specified in the LIBPATH= statement. Erase all emulators except the currently used one
- If you use an unsupported emulator:
  - Find out under which name the HLLAPI is to be called
  - VisualLift invokes PCSHLL.DLL at entry point of HLLAPI
  - If the host emulator has another DLL name and/or entry point, direct VisualLift to these by specifying the environment variables
    fclattachdll=DLL name
    fclattachproc=Procedurename

The application cannot be started

Solution: The initiation conditions are not fulfilled or
the specified host session is not operable or does not exist.

More information: **No connection to host emulator** (see previous symptom).

Would like to view the host application

Solution: To view the host application in the emulator session parallel to the Visual*Lifted* application, either specify the FCL_SESSMODE=SHOW variable in the CONFIG.SYS or specify the /FCL_SESSMODE=SHOW option for the application's program object (see page 66.)

IBM service asks for a trace

Solution: VisualLift can be started with a debug option. See Appendix C, “Options for Running a VisualLifted Application” on page 103.

VisualLift Windows RTE Symptoms

Main storage too small

Solution: To run a VisualLift application the required real storage size is application file size *plus* 6 MB. If the real storage is smaller, the application will not execute.

Rule of thumb: the recommended 10MB main storage will result in good performance if Visual*Lifted* application file size is smaller than 3MB.

Application not found

Solution: Check whether these two files are available:

The Visual*Lifted* application file (either, for example MYAPPL.AST or MYAPPL.AS_) is located in the directory *fclroot*\FCL\ASDATA
The associated message file is located in the directory
fclroot\FCL\NLSDATA\ENGLISH

DLLs not found when starting application

Solution:

- Make sure VisualLift is completely and correctly installed
- Make sure that you start the Visual*Lifted* application from the program manager
- Check the *working directory* entry for the program item. It should point to the directory where VisualLift resides.
- Check whether the environment variables FCLROOT and FCLRTS are set - VisualLift recommends *not* to set the environment variables. If they are set check whether they point to the VisualLift directories.

No connection to host emulator

Solution:

- Make sure the host emulator is started and a session is established *before* you start the Visual*Lifted* application
- Check whether the correct *session ID* is specified (in most cases *session A*)
- The directory where the host emulator code resides *must* be in the current path.
- If you have more than one host emulator or multiple versions of one emulator installed, make sure that only the correct one (but not the other ones) is specified in the PATH= statement. Erase all emulators except the currently used one
- If you use an unsupported emulator:
  - Find out under which name the HLLAPI is to be called
  - VisualLift invokes PCSHLL.DLL at entry point of HLLAPI
  - If the host emulator has another DLL name and/or entry point, direct VisualLift to these by specifying the environment variables
    fclattachdll=DLL name
    fclattachproc=Procedurename

The application cannot be started

Solution: The initiation conditions are not fulfilled or the specified host session is not operable or does not exist.

More information: **No connection to host emulator** (see previous symptom).

The old version of an application is invoked

Solution: If the Visual*Lifted* application is available in compressed format (for example, MYAPPL.AS_) and in un-compressed format (MYAPPL.AST) VisualLift invokes the compressed one, independent of the creation date. To invoke the actual Visual*Lifted* application, either delete the (old) compressed format or create a new Visual*Lifted* application in compressed format.

Would like to view the host application

Solution: To view the host application in the emulator session parallel to the Visual*Lifted* application do the following:

- Exit Windows
- Enter the command SET FCL_SESSMODE=SHOW
- Restart Windows

After leaving VisualLift, a VisualLift icon remains on the screen

Solution: This is a regular end situation - the icon indicates that the VisualLift libraries remain in memory for a fast start next time you invoke a Visual*Lifted* application.
If you are short on resources, close the icon to free the resources for other applications. Next time you start a Visual*Lifted* application the libraries will be loaded again.

VisualLift beeps and/or ends without an error message

Solution: Inspect the trouble log. It is named FCLMSG.LOG and is located in the directory *fclroot*\FCL
Use any editor or the NOTEPAD.EXE. The latest message is located at the bottom of the trouble log.

General protection fault in a DLL belonging to the host emulator program

Solution: In a DLL belonging to the host emulator program a protection fault occurred. Make sure you have the newest service level of the host emulator program.

Problem during installation of VisualLift Windows RTE

Solution: The software installer holds an INI-file named EPFWIS.INI in your windows directory. In case of problems edit the file and remove the section labeled `EPFINST_VisualLift`.

Try to install VisualLift again.

Problems using the RUMBA emulator

Solution: The directory where the file EEHLLAPI.DLL is located must be in the PATH= statement, for example
C:\RUMBA\SYSTEM

RUMBA does not automatically assign a session ID to your host session.

- Select the menu item *Session*
- Select the pull down *EHLLAPI Configuration*
- Select the check box *EHLLAPI SDK*
- Specify a session ID.

It is not necessary to set the environment variable *fclattachdll=* to the name used by RUMBA (EEHLLAPI). VisualLift automatically performs this task.

Workstation windows are too large

Solution: The monitor resolution has to be identical for Visual*Lifting* a host application and running a Visual*Lifted* application. Different monitor resolution may result in a unpredictable window layout. Keep in mind:

- When developing a Visual*Lifted* application on a workstation with higher resolution monitors (for example, XGA or SVGA) the Visual*Lifted* application runs on workstations with higher resolution monitors. Running the same Visual*Lifted* application on workstations equipped with VGA produces windows which might be larger than the physical screen. This has to be taken into consideration especially for the VisualLift RTE for Windows, because the window cannot be scrolled.
- When developing a Visual*Lifted* application on a workstation with VGA, the Visual*Lifted* application will run on a workstation with VGA and XGA/SVGA.

IBM service asks for a trace

Solution: VisualLift RTE can be started with a debug option. This option creates the file HLLAPI.LOG in the directory where VisualLift RTE is installed. This file is an ASCII file. The content is for internal IBM debugging purposes. To create this file invoke VisualLift RTE with the command line parameter `/DEBUG=EHLLAPI`.
For example, the invoke command is:

```
VLW appl A /DEBUG=EHLLAPI
```

Message in trouble log if screen is sizeable

Solution: Within VisualLift RTE for Windows the windows are not sizeable. If a sizeable screen is used within a Visual*Lifted* application, a message is written into the trouble log. This message can be suppressed. To suppress the message invoke VisualLift RTE with the command line parameter `/NSM`
For example, the invoke command is: `VLW appl A /NSM`

Appendix A. System Requirements

Supported Host Operating Systems

All versions and releases of:

- MVS/ESA
- VSE/ESA
- VM/ESA
- OS/390

VisualLift Application Development Environment

CPU	80386 and higher (80486 recommended)
Main Storage	12MB (16MB recommended)
Hard Disk Space	10MB (including RTE)
Monitor Resolution	VGA (XGA recommended)
Host Connection Hardware	327x adapter card, supported by host connection software
Pointing Device	PS/2 mouse (or equivalent)
Operating System	OS/2 2.1 and higher
Host Connection Software	• Communications Manager/2 1.0 and higher (or equivalent) • IBM Personal Communications/3270 for OS/2 Version 4 • Attachmate EXTRA! for OS/2 Version 1.2 or Version 2.0 • WallData RUMBA 3.0 for OS/2 • DCA IRMA 2.1 for OS/2

VisualLift OS/2 Run-Time Environment

CPU	80386 and higher (80486 with at least 50 MHz recommended)
Main Storage	12MB (16MB recommended)
Hard Disk Space	6MB
Monitor Resolution	VGA (XGA recommended)
Host Connection Hardware	327x adapter card, supported by host connection software
Pointing Device	PS/2 mouse (or equivalent)
Operating System	OS/2 2.1 and higher
Host Connection Software	• Communications Manager/2 1.0 and higher (or equivalent) • IBM Personal Communications/3270 for OS/2 Version 4 • Attachmate EXTRA! for OS/2 Version 1.2 or Version 2.0 • WallData RUMBA 3.0 for OS/2 • DCA IRMA 2.1 for OS/2

VisualLift Windows Run-Time Environment

CPU	80386 and higher (80486 with at least 50 MHz recommended)
Main Storage	6MB (10MB recommended)
Hard Disk Space	6MB
Monitor Resolution	VGA (XGA recommended)
Host Connection Hardware	As supported by the host connection software (see below)
Pointing Device	PS/2 mouse (or equivalent)
Operating System	IBM-DOS 5.0 and higher (recommended 6.3) or MS-DOS 5.0 and higher (recommended 6.2) • Windows 3.1 VisualLift requires Windows Extended 386 mode. • Windows for Workgroups 3.11 • Windows95 • Windows/NT is planned to be supported at a later point in time. Refer to README.WRI for additional information VisualLift supports Windows native. WINOS2 under OS/2 version 2 is *not* supported. Windows 3.1 under OS/2 Warp is supported for test purposes only. (The appearance may not be identical with native Windows). When using OS/2 use the OS/2 version of VisualLift.
Host Connection Software for Windows 3.1 and Windows for Workgroups 3.1.1 For Windows95 refer to README.WRI for info on supported emulators	Personal Communication/3270 3.1 for Windows and higher (or equivalent with EHLLAPI support) EXTRA! FOR WINDOWS 4.01 RUMBA 4.0 for Windows DCA IRMA WorkStation for Windows Version 2 NetSoft DynaComm/Elite Version 3.5 NCP 3270 Emulation for Windows, Version 2.30 by NCP engineering, Nürnberg

Optional Workstation Software

The coding of application provided routines requires the following software for OS/2:

- IBM Developer's Toolkit for OS/2 2.1 and higher
- IBM C++/2 Compiler 2.1 and higher (32 Bit)

The coding of application provided routines requires the following software for Windows:

- Microsoft Windows SDK 3.1 and higher
- Microsoft C Compiler 6.0 and higher (or equivalent).

Appendix B. System Information

Files of a VisualLifted Application

The following files make up a Visual*Lifted* application:

Figure 21. The VisualLift File Types Related to an Application

Extension	Purpose
***.AST**	The VisualLift application file - the Visual*Lifted* windows as well as the invocation and termination condition are contained within that file. It resides in *fclroot*\FCL\ASDATA
***.MSG**	The VisualLift application message file - text strings of the Visual*Lifted* application are contained within this file. It resides in *fclroot*\FCL\NLSDATA\ENGLISH
***.EVT**	(Optional) the event file - used to define events of a Visual*Lifted* application to be used by the RTE. It resides in *fclroot*\FCL\NLSDATA\ENGLISH
***.HLP**	(Optional) the help files for a Visual*Lifted* application. They resides in *fclroot*\FCL\NLSDATA\ENGLISH
***.DLL**	(Optional) the file where the application provided routines are located. They reside in *fclrts* \RTS\DLL For the Windows RTE they reside in *fclroot*
***.BMP**	(Optional) the bitmap files- used to incorporate pixel graphic into the PWS Window. They reside in *fclroot*\FCL\NLSDATA\ENGLISH
***.ICO and *.WCO**	(Optional) the icon files - used to incorporate icons into the PWS Window. They resides in *fclroot*\FCL\NLSDATA\ENGLISH
***.MET**	(Optional) the metafiles- used to incorporate vector graphic into the PWS Window. They reside in *fclroot*\FCL\NLSDATA\ENGLISH

Files of a VisualLifted Window

The following files make up a Visual*Lifted* window:

Figure 22. The VisualLift File Types Related to a Window

Extension	Purpose
***.AST**	The VisualLift PWS Window file - the definition of controls as well the mapping information are contained within this file. It resides in *fclroot*\FCL\ASDATA
***.MSG**	The VisualLift PWS Window message file - text strings of the PWS Window are contained within this file. It resides in *fclroot*\FCL\NLSDATA\ENGLISH
***.NLS**	(Optional) the national language support file - this file contains the extracted text strings of a PWS Window to be translated separately. It resides in *fclroot*\FCL\NLSDATA
***.SCN**	(Optional) the VisualLift host screen file - the scanned host screen belonging to the PWS Window is contained within this file. It resides in *fclroot*\FCL\HOSTSCAN
***.UIT**	(Optional) the VisualLift PWS Window (Text) file. It resides in *fclroot*\FCL\PWSTEXT

CONFIG.SYS Updates by OS/2 VisualLift

Take the case VisualLift is to be installed on `Drive D:\LIFT`
The CONFIG.SYS update performed by VisualLift are as follows:

```
LIBPATH=.;D:\LIFT\RTS\DLL;D:\LIFT\ADS\DLL;
SET PATH=.;D:\LIFT\RTS\BIN;D:\LIFT\ADS\BIN;
SET DPATH=.;D:\LIFT\RTS\HLP;D:\LIFT\ADS\HLP;
SET HELP=D:\LIFT\RTS\HLP;D:\LIFT\ADS\HLP;D:\LIFT\FCL\NLSDATA\ENGLISH;
SET INCLUDE=D:\LIFT\FCL\USEREXIT;
SET LIB=D:\LIFT\FCL\USEREXIT;
SET BOOKSHELF=D:\LIFT\FCL\NLSDATA\ENGLISH;D:\LIFT\ADS\HLP;

SET FCLROOT=D:\LIFT
SET FCLRTS=D:\LIFT
SET FCLADS=D:\LIFT
```

Environment Variables

The following environment variables are available within VisualLift:

Figure 23. The Environment Variables of VisualLift

Environment Variable	Description
SET FCLROOT	Specifies the drive and directory of the workstation where VisualLift resides.
SET FCLADS	Specifies the drive and directory of the workstation where the VisualLift application development environment is located.
SET FCLRTS	Specifies the drive and directory of the workstation where the VisualLift run-time environment is located.
SET FCLROOTLAN	Specifies the drive and directory of the server workstation where VisualLift is located.
SET FCLATTACHDLL (see note below)	Specifies the DLL offering the EHLLAPI interface. Specify the DLL name only without drive, directory, or extension.
SET FCLATTACHPROC (see note below)	Specifies the procedure offering the EHLLAPI interface.
SET FCLLANGUAGE	Specifies the language of VisualLift (default is English).
SET FCL_SESSMODE=SHOW	Specifies that the emulator session is visible at the same time as the Visual*Lifted* application. **Note:** if you are using the DynaComm and NCP 3270 emulators, the emulator session is always visible regardless if this variable is specified.
SET FCL_TIMEOUT	Specifies the time in seconds the Visual*Lifted* application waits for the host reaction.
SET FCL_EMULATOR	Specifies the supported emulator (OS/2 only). Allowed values are CM2, RUMBA, EXTRA!, IRMA, and PC3270.

Note: For VisualLift RTE for Windows, FCLATTACHDLL and FCLATTACHPROC have to be set only if a non-supported emulator is used.

Environment Variables and Host Emulators for VisualLift RTE for Windows

For VisualLift RTE for Windows, the following default values for supported emulators have been implemented:

Figure 24. The Environment Variables for Supported Emulators

Emulator	FCLATTACHDLL	FCLATTACHPROC
PC/3270	PCSHLL	HLLAPI
EXTRA!	PCSHLL	HLLAPI
Rumba	EEHLLAPI	HLLAPI
IRMA	ACS3EHAP	HLLAPI
DynaComm	HLLAPI	HLLAPI
NCP 3270	ACS3EHAP	HLLAPI

Appendix C. Options for Running a VisualLifted Application

This part lists the options which can be specified for running a Visual*Lifted* application.

`Options`: (optional) can be one of the following:

[/P=pos] Overrides the position of the invocation string defined in the application settings.

[/S=invocstring] Invocation string that overrides the invocation string defined in the application settings. If the invocation string contains blanks it should be enclosed in single quotes or double quotes. If the invocation string contains single/double quotes, then the string must be enclosed in double/single quotes.

[/IS=initstring] Overrides the initiation condition string in the application settings. Only one initiation condition may be specified.

[/IP=initpos] Overrides the initiation condition position defined in the application settings.

[/Q] If this option is specified, the text of the host emulator window title remains unchanged if a host screen is not recognized at run-time. The default behavior is that the text of the emulator window is changed to *VisualLift - X: Screen not recognized (applname)*

[/FCLROOT=path] Overrides the setting of the environment variable FCLROOT for this invocation. FCLROOT specifies the path where VisualLift resides.

[/FCLROOTLAN=path] Overrides the setting of the environment variable FCLROOTLAN for this invocation. FCLROOTLAN specifies the path of the LAN server workstation where VisualLift resides.

[/FCLRTS=path] Overrides the setting of the environment variable FCLRTS for this invocation. FCLRTS specifies the path where the VisualLift run-time environment resides.

[/FCLLANGUAGE=language]
Overrides the setting of the environment variable FCLLANGUAGE for this invocation. FCLLANGUAGE specifies the language of VisualLift. The default language is English.

[/FCL_SESSMODE={HIDE|SHOW}]
Overrides the setting of the environment variable FCL_SESSMODE for this invocation. FCL_SESSMODE specifies if the emulator window is not visible/visible at the same time as the Visual*Lifted* application.

[/FCL_TIMEOUT=timeoutval]
Overrides the setting of the environment variable FCL_TIMEOUT for this invocation. FCL_TIMEOUT species the time in seconds the Visual*Lifted* application waits for a host reaction.

[/FCLATTACHDLL=dllname]
Overrides the setting of the environment variable FCLATTACHDLL for this invocation. Specifies the DLL name only without drive, directory, or extension.

[/FCLATTACHPROC=procname]

Overrides the setting of the environment variable FCLATTACHPROC for this invocation. FCLATTACHPROC specifies the procedure offering the EHLLAPI interface.

[/FCL_EMULATOR={CM2|PC3270|IRMA|RUMBA|EXTRA!|EMUTEST}]

Overrides the setting of the environment variable FCL_EMULATOR for this invocation. FCL_EMULATOR specifies one of the supported emulators (this does not apply for VisualLift RTE for Windows).

[/DEBUG=debugoption, debugoption]

Creates various traces for debugging purposes. Debug options are ANALYZE, PFE, TRACE, VT, PSDUMP, and EHLLAPI (for VisualLift RTE for Windows only EHLLAPI is supported).

Glossary

Glossary terms are defined as used within this book. If you cannot find the term you are looking for, refer to the index or to:

- *Object-Oriented Interface Design, SC34-4399*
- *IBM Dictionary of Computing*, New York: McGraw-Hill, 1994.

Terms of this glossary are also available in the help information of VisualLift and in the VisualLift Reference.

A

action routine. An action routine is used to affect the Visual*Lifted* user interface. For example, controls within a workstation window may depend on another control without exchanging data with the host application.

ADE. The Application Development Environment (ADE) is the part of VisualLift where the new user interface for the host application is designed, created and bundled. The ADE becomes visible in the form of the VisualLift Workbench. The VisualLift Workbench consists of a set of object oriented tools.

application. A VisualLift application represents the new user interface for a host application. Throughout VisualLift a VisualLift application is called application. An application contains the created PWS Windows as well as invocation and termination information of the host application. The application is the result of Visual*Lifting*.

application provided routines. A Visual*Lifted* application may provide additional functions by supplying user-written routines. Within these routines, the set of functions provided by the run-time environment may be extended. The Visual*Lifted* application has to provide one or more DLLs containing the routines. It also has to define the routines to be called for a specific window.

B

button. (1) A mechanism on a pointing device, such as a mouse, used to request or initiate an action or a process. (2) A graphical device that identifies a choice. (3) A graphical mechanism that, when selected, performs a visible action. For example, when a user clicks on a list button, a list of choices appears.

C

cascaded menu. A menu that appears from, and contains choices related to, a cascading choice in another menu. Cascaded menus are used to reduce the length of a pull-down menu or a pop-up menu.

cascaded choice. A choice on a menu that leads to a cascaded menu containing related choices. A cascading choice is indicated by a rightward-pointing arrow (→) to the right of the choice.

check box. A square box with associated text that represents a choice. When a user selects the choice, an **x** (or a checkmark) appears in the check box to indicate that the choice is selected. The user can clear the check box by selecting the choice again, thereby canceling the selection. Check boxes may be used in a group to provide a multiple choice field.

checking routine. A checking routine is used to extend the input validation provided by the run-time environment. A checking routine acts on the user interface of the workstation without exchanging data with the host application.

check marks. A character that indicates that a settings choice is active.

click. To press and release a button on a pointing device without moving the pointer of the object or choice.

combination box. A control that combines the functions of an entry field and a list box. A combination box contains a list of objects that a user can scroll through and select from to complete the entry field. Alternatively, a user can type text directly into the entry field.

controls. Visual user-interface components that allow a user to interact with data. Controls are usually identified by text, for example, headings, labels in push buttons, field prompts, and titles in windows.

current-setting indicator. A visible indication that a choice is active or inactive, for example the **x** (or a checkmark) that appears in a check box when it is selected.

de-selection. The process of removing selection from a previously selected object.

D

double-click. To press and release a button on a pointing device twice while a pointer is within the limits that the user has specified for the operating environment.

drag. To use a pointing device to move or copy an object. For example, a user can drag a window border to make it larger. To drag something, a user presses and holds a button on the pointing device while moving the pointing device.

drag and drop. To directly manipulate an object by moving it and placing it somewhere else using a pointing device.

drop-down combination box. A combination box in which the list is hidden until a user takes an action to make it visible. A drop-down combination box contains a list of objects or settings choices that a user can scroll through and select from to complete the entry field. Alternatively, a user can type text directly into the entry field. The typed text does not have to match one of the objects or settings choices contained in the list.

drop-down list. A drop-down list is a variation of a list box. A drop-down list only displays one item until the user takes an action to display the other objects or choices. Like a list box, the drop-down list does not allow a user to type information into it.

E

emphasis. Highlighting, color change, or other visible indication of the condition of an object or choice and the effect of that condition on a user's ability to interact with that object or choice. Emphasis can also give a user additional information about the state of an object or choice.

entry field. A control into which a user places text. Its boundaries are usually indicated. Entry fields can scroll if more information is available than is currently visible.

event. An event is a condition at the host that is not covered by data in the application description file. These events are specified globally in event definition files. Event definitions can be added to the system event definition file.

G

graphic editor. With the graphic editor controls can be created or changed via dialogs and drag and drop operations.

group box. A rectangular box drawn around a group of controls to indicate that the controls are related and to provide a label for the group.

group heading. A heading that identifies a set of related fields.

H

help exit routine. A help exit routine is used to provide context sensitive help information for an application.

host screen. A Host Screen object is the screen capture of a host screen. The Host Screen object is located at the workstation. It is the base for the design of PWS Windows. It is also used to define the mapping between Host and PWS when working with the VisualLift graphic editor.

I

icon. A pictorial representation of an object, consisting of an image, image background, and a label. Small icons can be substituted for regular icons. The small icon in the title bar of a window is another icon for the object that is displayed in the window.

information window. A specific part of a window in which information about the object or choice that the cursor is on is displayed. The information area can also contain a message about the completion of a process.

L

list box. A control that contains a list of objects or settings choices that a user can select. List boxes support single or multiple selection.

M

mapping. The connection between the input/output fields on the host screen and the designed workstation controls. The mapping ensures that the information received and sent to/from the host application is handled by the corresponding workstation control.

mapping routine. A mapping routine is used to extend the default Host-PWS mapping of the run-time environment.

menu. A list of choices that can be applied to an object. A menu can contain choices that are not available for selection in certain contexts. Those choices are indicated by reduced contrast.

menu bar. The area near the top of a window, below the title bar and above the rest of the window, that contains routing choices that provide access to pull-down menus. Typically a menu bar choice is a single word.

menu bar choice. A graphical or textual item on a menu bar that provides access to pull-down menus which contain choices that can be applied to an object.

menu choice. A graphical or textual item on a menu. A user selects a menu choice to work with an object in some way.

message. Information not requested by a user but displayed by a product or application in response to an unexpected event, or when something undesirable could occur, or as information.

mouse. A commonly used pointing device that has one or

more buttons that a user presses to interact with a computer system.

mouse button. A mechanism on a mouse pointing device used to select choices, initiate actions, or manipulate objects with the pointer. The button makes a *clicking* sound when pressed and released.

multi-line entry field. A control into which a user places several lines of text. Its boundaries are usually indicated. Multi-line entry fields can be scrolled if more information is available than is currently visible.

multiple selection. A type of selection in which a user can select any number of objects or settings choices, or not select any.

N

notebook. A graphical representation that resembles a bound notebook that contains pages separated into sections by tabbed divider pages. A user can turn the pages of a notebook to move from one section to another.

O

object. An item that can be manipulated as a unit and that a user works with to perform a task. An object can be represented as text, image, graphic, or audio.

P

padding. Padding is the filling up of characters of entry fields during mapping of Host ⇔ PWS controls. Whenever less characters than the maximum for the host value are specified, padding takes effect.

padding character. The character used to fill up missing characters of a host entry field if less characters than the maximum are specified. A padding character may be either any character or Null (hexadecimal zero '00'X). Blank is the default.

panel identification exit routine. A panel identification exit routine is used to modify the results of the panel identification process according to the needs of the application.

pointer. A visible cue, usually in the shape of an arrow, that a user can move with a pointing device. Users place the pointer over objects they want to work with.

pointing device. A device, such as a mouse, trackball, or joystick, used to move a pointer on the screen.

pop-up menu. A menu that, when requested, is displayed next to the object it is associated with. It contains choices appropriate for a given object or set of objects in their current context.

primary window. An area on the display screen used to present a view or to conduct a dialog with a user. The two types of windows are primary windows and secondary windows. Windows are used to present controls, messages, and help information.

programmable workstation (PWS). A workstation that has some degree of processing capability and that allows a user to change its functions.

progress indicator. One or more controls used to inform a user about the progress of a process.

project. A Project within VisualLift is the place where the Visual*Lifting* takes place. Define Projects to organize all objects of one Application within one Project. Projects available reside in the VisualLift Workbench. Within a Project the following objects may reside: Application, Host Screen, PWS Window, and PWS Window (Text).

pull-down menu. A menu that extends from a selected choice on a menu bar or from the system-menu symbol. The choices in a pull-down menu are related to one another in some manner.

push button. A button, labeled with text, graphics, or both, that represents an action that will be initiated when a user selects it.

PWS. Programmable workstation.

PWS Window. A PWS Window is the graphical representation of a PWS Window (Text). The PWS Window contains window and control definitions. It can be created or modified with the VisualLift graphic editor. A PWS Window can be converted into a PWS Window (Text) and vice versa. PWS Windows are the source for building an Application. There are two representations of PWS Windows:


PWS Window without inserted panel ID.


PWS Window with inserted panel ID.

PWS Window (Text). A PWS Window (Text) is the textual representation of a PWS Window. The PWS Window (Text) contains the window and control definitions in a declarative language. It can be created or modified with an editor. A PWS Window (Text) can be converted into a PWS Window and vice versa.

R

radio button. A control used to display mutually exclusive textual settings choices visualized by a circle with text beside it. Radio buttons are combined to show a user a fixed set of choices from which the user can select one. The circle becomes partially filled when a choice is selected.

RTE. The Run-Time Environment (RTE) is the executing part of VisualLift. It is used to run the Visual*Lifted* application. The RTE is the bridge between the user interface on the workstation and

the application on the host. Basically the RTE invokes and terminates the host application, displays the workstation user interface, and manages user input.

S

scroll bar. A window component that shows a user that more information is available in a particular direction and can be scrolled into view. Scroll bars can be either horizontal or vertical.

secondary window. A window that contains information that is dependent on information in a primary window and is used to supplement the interaction in the primary window.

selected-state emphasis. Emphasis used on a choice or object to indicate that it is selected.

selection. The act of explicitly identifying one or more objects to which a subsequent choice will apply.

shortcut key. A key or a combination of keys assigned to a menu choice that initiates that choice, even if the associated menu is not currently displayed.

slider. A visual component of a user interface that represents a quantity and its relationship to the range of possible values for that quantity. A user can also change the value of the quantity.

spin button. A control used to display, in sequence, a ring of related but mutually exclusive choices. It contains a field that can accept user input, which allows a user to make a selection by typing a valid choice, or a field that can display a value that the user can merely accept. The user can change the value by spinning through the ring of choices.

status area. A part of a window where information appears that shows the state of an object or the state of a particular view of an object.

T

tabbed divider page. A graphical representation of a tabbed page in a notebook. Tabbed divider pages separate sections of the notebook.

template. An object that you can use as a model to create additional objects. When you drag a template you create another of the original object by dropping it over the target destination. VisualLift offers template objects for Application, Project, Host Screen, PWS Window, and PWS Window (Text).

trouble log. The trouble log is a file where error messages occurring in the run-time environment are recorded. For each message a detailed explanation can be requested. The purpose of the trouble log is to assist users in problem determination.

U

unavailable-state emphasis. A visible cue that indicates that a choice cannot be selected.

user control. Used to reserve space in a window for a user defined control. The user defined control can be anything. A window procedure has to be provided by the application to manage the user control.

V

value set. A control that allows a user to select one choice from a group of mutually exclusive choices. A value set is used primarily for graphical choices.

variable table. A VisualLift data structure containing the definition of all the variables used in a function to communicate with the user.

VisualLift. A tool to modernize the user interface of existing host applications. The new user interface is located on the workstation. The host application remains untouched. This means that an existing host application has now two user interfaces: the existing one and the Visual*Lifted* one.

W

window. An area with visible boundaries that presents a view of an object or with which a user conducts a dialog with a computer system.

window procedure. A window procedure for private controls is used to define controls which are not supported by VisualLift. Window procedures for private controls can be specified as a presentation attribute of a user control.

window title. The area on a title bar that contains the name of the object or a short description of the contents of the window.

Index

Numerics

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

File Number: S370/S390-20
Program Number: 5648-109

Printed in U.S.A.

SC33-6691-02

Table Definitions

id	File	Page	References
REQA	FCLAREQ	97	97, 97, 98
SYSEXT	FCLASYS	99	99, 100
ENVIR	FCLASYS	101	101
ENVV	FCLASYS	102	102

Figures

id	File	Page	References
SKILL	FCLAINTR	3	2 3
HOST	FCLAWHAT	5	3 5
VL	FCLAWHAT	5	4 5
BASE	FCLACONC	7	5 7
TASK1	FCLACONC	8	6 8, 9
TASK2	FCLACONC	11	7 10
3270	FCLACONC	11	8 11
VLAN	FCLACONC	19	9 19
HIER	FCLASOL	25	10
SOLG	FCLASOL	28	11
SOLE	FCLASOL	29	12
ENTRYM	FCLASOL	30	13
COMP1	FCLATATH	32	14
TASK	FCLATATH	33	15 31, 31
COMP	FCLATATH	38	16
TCB1	FCLATATH	48	17
TCB2	FCLATATH	48	18
SMPTXT	FCLATATH	50	19 49
TLE	FCLAPROB	88	20
SYSAPP	FCLASYS	99	21
SYSWIN	FCLASYS	100	22
ENVIRM	FCLASYS	101	23
ENVVA	FCLASYS	102	24

Headings

Index Entries

IPROBL	FCLAPROB		
		87	(1) common problems 89, 93
ITR	FCLAPROB		
		88	(1) problem (2) trouble log 88, 88
ITROU	FCLAPROB		
		88	(1) trouble log 88, 88
IREQ	FCLAREQ		
		97	(1) system requirements 97, 97, 98, 98
IPP	FCLASYS		
		99	(1) application provided files 100

Footnotes

id	File	Page	References
PATH	FCLAINTR		
		4	1 3
VIS	FCLAINTR		
		4	2 4

Revisions

id	File	Page	References
1	FCLAG SCRIPT		
		109	1, 1, 2, 3, 7, 7, 9, 9, 14, 14, 15, 15, 15, 15, 15, 15, 16, 16, 16, 16, 17, 17, 17, 17, 17, 17, 17, 17, 18, 18, 18, 18, 19, 19, 19, 19, 19, 20, 20, 20, 20, 20, 23, 23, 23, 23, 24, 24, 25, 25, 26, 26, 26, 26, 27, 27, 27, 27, 28, 28, 28, 28, 29, 29, 29, 29, 29, 29, 30, 30, 32, 32, 34, 34, 36, 36, 36, 36, 36, 36, 36, 36, 37, 37, 38, 38, 38, 38, 39, 39, 39, 39, 41, 41, 41, 41, 42, 42, 42, 42, 44, 44, 44, 44, 46, 46, 46, 46, 46, 46, 47, 47, 47, 47, 48, 48, 48, 48, 48, 48, 48, 48, 49, 49, 49, 49, 53, 53, 53, 53, 53, 53, 54, 54, 54, 54, 54, 54, 54, 54, 55, 55, 56, 56, 56, 56, 58, 58, 58, 58, 59, 59, 60, 60, 61, 61, 61, 61, 61, 61, 62, 62, 63, 63, 63, 63, 63, 63, 63, 63, 64, 64, 65, 65, 66, 66, 67, 67, 67, 67, 68, 68, 68, 68, 71, 72, 74, 74, 75, 75, 77, 77, 79, 79, 82, 82, 84, 84, 84, 85, 85, 85, 85, 85, 85, 85, 90, 91, 92, 92, 92, 92, 92, 92, 92, 92, 94, 94, 97, 97, 97, 97, 97, 97, 99, 99, 99, 99, 99, 99, 100, 100, 103, 103, 103, 103, 103, 103, 103, 103, 103, 103, 103, 103, 103, 103, 103, 103, 104, 104, 104, 104, 107, 107, 107, 107
2	FCLAG SCRIPT		
		109	ii, ii, 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 6, 6, 6, 6, 9, 10, 12, 18, 19, 21, 23, 24, 27, 30, 31, 31, 31, 32, 34, 34, 36, 36, 36, 36, 36, 36, 36, 36, 38, 38, 39, 39, 41, 41, 41, 41, 42, 42, 42, 42, 42, 42, 42, 42, 43, 43, 43, 43, 45, 45, 45, 45, 46, 46, 47, 47, 47, 47, 48, 49, 49, 49, 53, 53, 53, 53, 53, 53, 54, 54, 54, 54, 55, 55, 56, 56, 56, 56, 58, 58, 58, 58, 58, 58, 58, 58, 58, 60, 60, 60, 60, 61, 61, 63, 63, 66, 66, 66, 71, 71, 71, 71, 82, 82, 83, 83, 84, 84, 87, 87, 87, 87, 89, 89, 90, 90, 90, 91, 95, 95, 97, 97, 104, 106, 106
3	FCLAG SCRIPT		
		109	2, 2, 9, 9, 25, 26, 34, 34, 36, 36, 38, 38, 39, 39, 40, 40, 42, 42, 44, 44, 47, 47, 48, 48, 48, 48, 48, 48, 53, 53, 53, 54, 57, 57, 57, 57, 57, 57, 59, 59, 61, 61, 63, 63, 63, 63, 63, 63, 63, 63, 64, 64, 64, 64, 64, 64, 64, 64, 64, 65, 65, 65, 66, 66, 67, 67, 67, 67, 67, 67, 68, 68, 68, 68, 68, 68, 68, 68, 103, 103, 103, 103, 103, 103
4	FCLAG SCRIPT		
		109	32, 32, 34, 34, 67, 67, 68, 68, 68, 68
5	FCLAG SCRIPT		
		109	

Spots

id	File	Page	References
PIPR	FCLACONC		
		14	(no text)
			68
SOLU	FCLASOL		
		28	(no text)
			1
STATA	FCLATATH		
		31	(no text)
			4
WBS	FCLATATH		
		34	(no text)
			91
HOID	FCLATATH		
		35	(no text)
			89
S2	FCLATATH		
		38	(no text)
DRADRO	FCLATATH		
		44	(no text)
			43
CTM	FCLATATH		
		48	(no text)
			45, 47
INCO	FCLATATH		
		58	(no text)
			89
INVOC	FCLATATH		
		59	(no text)
			91
APEND	FCLATATH		
		60	(no text)
			90, 90
OPT	FCLATATH		
		66	(no text)
			91, 92
PROPA	FCLATATH		
		66	(no text)
			89
LOCK	FCLAWINS		
		76	(no text)
			77, 81

Processing Options

Runtime values:

Document fileid	FCLAG SCRIPT
Document type	USERDOC
Document style	IBMXAGD
Profile	EDFPRF40
Service Level	0030
SCRIPT/VS Release	4.0.0
Date	98.02.05
Time	15:00:19
Device	PSA
Number of Passes	4
Index	YES
SYSVAR D	YES
SYSVAR X	YES

Formatting values used:

Annotation	NO
Cross reference listing	YES
Cross reference head prefix only	NO
Dialog	LABEL
Duplex	YES
DVCF conditions file	(none)
DVCF value 1	(none)
DVCF value 2	(none)
DVCF value 3	(none)
DVCF value 4	(none)
DVCF value 5	(none)
DVCF value 6	(none)
DVCF value 7	(none)
DVCF value 8	(none)
DVCF value 9	(none)
Explode	NO
Figure list on new page	YES
Figure/table number separation	NO
Folio-by-chapter	NO
Head 0 body text	(none)
Head 1 body text	Chapter
Head 1 appendix text	Appendix
Hyphenation	NO
Justification	NO
Language	ENGL
Keyboard	395
Layout	OFF
Leader dots	YES
Master index	(none)
Partial TOC (maximum level)	4
Partial TOC (new page after)	INLINE
Print example id's	NO
Print cross reference page numbers	YES
Process value	(none)
Punctuation move characters	.,
Read cross-reference file	(none)
Running heading/footing rule	NONE
Show index entries	NO
Table of Contents (maximum level)	3
Table list on new page	YES
Title page (draft) alignment	RIGHT
Write cross-reference file	(none)

Imbed Trace

Page i	FA299EDN
Page vii	FCLANOT
Page x	FCLAINDX
Page x	FCLAINTR
Page 4	FCLAWHAT
Page 6	FCLACONC
Page 22	FCLAINST
Page 24	FCLASOL
Page 30	FCLATATH
Page 70	FCLAWINS
Page 77	FCLABAT
Page 80	FCLAHINT
Page 85	FCLAPROB
Page 96	FCLAREQ
Page 98	FCLASYS
Page 102	FCLAOPT
Page 105	FCLAGLOS