Professional Series Portability Guide

Version 9.3

Printed: December 2009

DCN PG-01-9301

Synergex
2330 Gold Meadow Way
Gold River, CA 95670 USA

http://www.synergex.com
phone 916.635.7300
fax 916.635.6549

Contents

Part 2: Developing for UNIX

3 Synergy/DE on UNIX: The Basics

4 Statements and Subroutines on UNIX

5 Terminal Characteristics on UNIX

Preface

About this manual

Synergy/DE™ (Synergy Development Environment™) is designed to be portable, but there are cases where functions, requirements, and processes differ on the various supported platforms. There are also differences and compatibility issues between the major versions of Synergy/DE. The goal of this guide is to document the functions, requirements, and processes that are unique to each of the platforms on which Synergy/DE runs.

The Professional Series Portability Guide is written for users who are already familiar with programming concepts and terminology (but not necessarily with Synergy Language) and are familiar with the platforms discussed in this guide.

Manual conventions

Throughout this manual, we use the following conventions:

- In code syntax, text that you type is in `Courier` typeface. Variables that either represent or should be replaced with specific data are in *italic* type.
- Optional arguments are enclosed in *[*italic square brackets*]*. If an argument is omitted and the comma is outside the brackets, a comma must be used as a placeholder, unless the omitted argument is the last argument in a subroutine. If an argument is omitted and the comma is inside the brackets, the comma may also be omitted.
- Arguments that can be repeated one or more times are followed by an ellipsis…
- A vertical bar (|) in syntax means to choose between the arguments on either side of the bar.
- Data types are **boldface**. The data type in parentheses at the end of an argument description (for example, (**n**)) documents how the argument will be treated within the routine. An **a** represents alpha, a **d** represents decimal or implied-decimal, an **i** represents integer, and an **n** represents numeric (which means the type can be **d** or **i**).
- To "enter" data means to type it (or highlight it, in the case of a selection window entry) and then press ENTER. ("ENTER" refers to either the ENTER key or the RETURN key, depending on your keyboard.)

WIN

Items or discussions that pertain only to a specific operating system or environment are called out with the name of the operating system.

Other useful publications

- *Getting Started with Synergy/DE*
- *UI Toolkit Reference Manual*
- *Synergy Language Reference Manual*
- *Synergy Language Tools*
- *Environment Variables and System Options*
- *Repository User's Guide*
- *ReportWriter User's Guide*
- Synergy/DE Release Notes
- *Migrating Your Application to Windows* (available on the Online Manuals CD)

Product support information

If you cannot find the information you need in this manual or in the publications listed above, you can call the Synergy/DE Developer Support department at (800) 366-3472 (in North America) or (916) 635-7300. To purchase Developer Support services, contact your Synergy/DE account manager at the above phone numbers.

Before you contact us, make sure you have the following information:

- The version of Synergy/DE product(s) you are running
- The name and version of the operating system you are running
- The hardware platform you are using
- The error mnemonic and any associated error text (if you need help with a Synergy/DE error)
- The statement at which the error occurred
- The exact steps that preceded the problem
- What changed (for example, code, data, hardware) before this problem occurred
- Whether the problem happens every time and whether it is reproducible in a small test program
- Whether your program terminates with a traceback, or whether you are trapping and interpreting the error

Synergex Professional Services Group

If you would like assistance implementing new technology or would like to bring in additional experienced resources to complete a project or customize a solution, Synergex™ Professional Services Group (PSG) can help. PSG provides comprehensive technical training and consulting services to help you take advantage of Synergex's current and emerging technologies. For information and pricing, contact your Synergy/DE account manager at (800) 366-3472 (in North America) or (916) 635-7300.

Comments and suggestions

We welcome your comments and suggestions for improving this manual. Send your comments, suggestions, and queries, as well as any errors or omissions you've discovered, to **doc@synergex.com**.

Part 1: Developing for Windows

This section of the *Professional Series Portability Guide* contains information on Synergy Language and UI Toolkit that is specific to Windows environments.

1

Synergy/DE on Windows: The Basics

Windows Terminology

We use the following terminology in our Windows documentation. You should be familiar with these terms before you read the rest of this and the other chapters in "Developing for Windows."

This list is meant to build on (not repeat) the explanations of Windows terminology found in Microsoft Windows manuals. If you are not already familiar with standard Windows terms (such as icon, scroll bar, minimize/maximize buttons, and so forth), refer to your Microsoft Windows documentation.

API

API, or Application Programming Interface, is a set of subroutines and functions used to communicate between different parts of a system. This is the level of the system that the application programmer uses. For example, UI Toolkit provides an API that can be called by a Synergy Language program; this API is the interface between the Synergy Language program and the UI Toolkit.

DLL

A DLL, or Dynamic Link Library, is a library file containing API routines. You can access these libraries at runtime without explicitly linking them to an application. Most (third-party) Windows applications provide and use their own DLLs.

Synergy/DE on Windows Components

The two primary components of Synergy/DE on Windows are

- Synergy Language
- UI Toolkit

Synergy Language

Synergy Language on Windows supports the same statements and subroutines that other Synergy Language products support, with some slight variations. For more information, see chapter 2, "Statements, Functions, and Subroutines on Windows."

The Windows version of the runtime is designed for use with UI Toolkit.

UI Toolkit

The UI Toolkit routines enable you to use native Microsoft Windows features such as menus, message boxes, combination (or combo) boxes, edit controls, status windows, scroll bars, buttons, and list boxes. Because Toolkit gives your application the ability to use most standard Windows features, we recommend that you take full advantage of these Toolkit subroutines.

For more details on how Toolkit works in a Windows environment, see chapter 2, "Statements, Functions, and Subroutines on Windows."

Windows Characteristics

Synergy/DE on Windows provides you with a graphical environment for developing applications. This graphical environment offers greater control of your applications than that offered by traditional UNIX and OpenVMS environments. You can run multiple applications simultaneously, allowing background applications to run while you use the foreground application. The number of multiple applications that can run simultaneously is limited only by your system's available memory.

Filenames

Synergy/DE does not fully support the long filenames of Windows in regard to spaces. The Synergy Language OPEN statement and the UI Toolkit U_OPEN subroutine support spaces in filenames; however, other Synergy/DE tools may truncate filenames at the first space.

What your application will look like

When a Synergy application starts in Windows, a native application window appears. A separate window is created when the Synergy debugger or the Toolkit window debugger is invoked. For more information, see "Using the debugger on Windows" in the "Debugging Your Synergy Programs" chapter of *Synergy Language Tools*.

Depending upon the screen resolution being used, the number of rows and columns being displayed, the font used for the application window, and the setting of the MINIMIZE_LEADING environment variable, when an application starts up it may require more space than the screen affords.

If so, it will be limited to the available space, and a vertical or horizontal scroll bar will be displayed to allow the user to scroll to the unseen area of the application window. Rather than enlarging the application window to add the scroll bar, the scroll bar occupies part of the initial size of the application window, in order to avoid unpleasant resizing of the application window upon startup or whenever the scroll bars are removed.

Because of this, however, it is likely that the addition of one scroll bar (for instance, the vertical one) will result in the addition of the other (for instance, horizontal) scroll bar, so that the user may view the portion of the application which was just occluded by the first scroll bar. To eliminate the second scroll bar, the user may stretch the application window along that scroll bar's axis until it disappears.

Big-endian and little-endian

Computer systems store integers in either big-endian or little-endian format. For big-endian systems, the low-order byte has the highest address. For little-endian systems, the low-order byte has the lowest address. Computers with Motorola 68*xxx* series chips and non-Compaq RISC chips are big-endian. Computers with Intel and Compaq chips are little-endian. See the table on page 3-4 for a list of the endian types of various systems.

Microsoft Windows environments are little-endian systems. Synergy applications developed on Windows environments are therefore portable across little-endian machines (such as SCO UNIX). The reverse is also true: you can run a Synergy application developed on an SCO UNIX machine in a Windows environment without re-creating the executables. To make your applications more Windows-like, however, there are changes that you need to make. For more information, see chapter 2, "Statements, Functions, and Subroutines on Windows."

Requirements

The following sections discuss the specific requirements for developing and running Synergy applications in a Windows environment.

Requirements for developing Synergy programs

You need to have installed Synergy/DE Professional Series to develop a Synergy program on Windows.

You may also choose to develop on other little-endian systems, such as SCO UNIX, instead of developing in a Windows environment, and then move to a Windows environment to run and test your application.

Development tools

You can run the Synergy/DE development tools from a command prompt window. This is important if you usually create batch files to build your executables.

Requirements for running Synergy programs

If you are using UI Toolkit subroutines, you need to have installed Synergy/DE Professional Series to run a Synergy program under Windows.

Running existing applications

Most existing Synergy applications will run without any additional work beyond installing and using the Windows runtime. Note the following:

- First check for any conditionally compiled code. If any exists, evaluate it for possible changes when running on Windows.

  If changes are necessary, you can use the compiler built-in identifier D_GUI to conditionalize your code. If you make any changes, you must recompile and relink your application.

  We recommend conditionalizing the code at runtime by comparing the *machine* argument returned from the ENVRN subroutine to the 101 system code. With this method, you can take advantage of little-endian system compatibility, which requires no recompiling or relinking.

- If your application has been compiled and linked using an older version of **dbl** and **dblink** (older than the versions running on Windows) or has been built on a big-endian system, you must recompile and relink your application.

Running existing UI Toolkit applications

If your application is a UI Toolkit application that runs on non-Windows *little-endian* systems and was compiled and linked with the same versions running on Windows, it will work with Synergy/DE on Windows. You will just need to do the following:

1. Copy any Toolkit window libraries from your other little-endian system to your Windows system.
2. Relink with the Toolkit's executable library (**tklib.elb**) if your application is currently linked with **tklib.olb**. If your application is already linked with the **.elb**, you do not need to relink on the Windows system.

If your application is a Toolkit application that runs on non-Windows *big-endian* systems and was compiled and linked with the same versions running on Windows, it will work with Synergy/DE on Windows. You will need to do the following:

1. Recompile and relink your application with the Toolkit's executable library (**tklib.elb**).
2. Rescript your window libraries on Windows.

For more information on recompiling and relinking, see chapter 2, "Statements, Functions, and Subroutines on Windows," in this manual and the "Building and Running Synergy Language Programs" chapter of *Synergy Language Tools*.

C interface requirements

The C interface enables you to call C routines from within your Synergy application. There are two methods for doing this:

- Rebuild the runtime (**dbr.exe**) to include the C routines, and then call them from your application.
- Create a DLL containing the C routines, and then use the DLL functions to access them.

Before you can use either of these methods, some requirements must be met. The following sections describe these requirements.

Rebuilding the runtime

To rebuild the Synergy runtime on Windows, you must have installed and configured a suitable software development environment; refer to the comments in the **makedbr.bat** file for specific products and versions.

Using a DLL

To use a DLL with Synergy/DE on Windows, you must

- create a 32-bit DLL. Refer to your Microsoft Windows documentation for more information.
- explicitly “export” the routines that your Synergy program will call.
- use the Synergy Language %DLL_*xxx* functions to access the routines in the DLL. Refer to the “Synergy DLL API” chapter of the *Synergy Language Reference Manual* for syntax specifications.

Using Initialization Settings

Initialization files define your Windows environment. Windows applications modify their operation according to users' requirements set in initialization files. Refer to your Microsoft documentation for more information on Windows initialization files.

Synergy.ini and **synuser.ini** are the initialization files that contain environment variables affecting the Synergy runtime and Synergy/DE development tools and executables on Windows. The **synergy.ini** file contains system- or application-specific settings, for multiple users. The **synuser.ini** file contains user-specific settings (for example, personal preferences such as colors, fonts, the state of the application window, the position and size of the print preview window, and any overrides to system-specific settings).

Refer to the "Environment Variables" chapter of *Environment Variables and System Options* for more information on Synergy initialization files and initialization settings syntax.

2

Statements, Functions, and Subroutines on Windows

This chapter discusses the differences between Windows and UNIX or OpenVMS environments when using Synergy Language and UI Toolkit.

Synergy Language on Windows

Though all versions of Synergy Language are portable, each operating system offers different features, imposes a few unique constraints, or requires some system-specific procedures.

This section identifies such specific features, constraints, and procedures for Windows environments. For more information, refer to the *Synergy Language Reference Manual*.

Synergy Language statements

ACCEPT, DISPLAY, and READS

The ACCEPT, DISPLAY, and READS statements should not be mixed with the W_*xxx* routines because the characters are displayed in a "console" window.

DETACH

The DETACH statement is not available.

F10 key

In Windows, the F10 key is a reserved key that activates the system menu. It will not be returned to your application when using ACCEPT or READS, but will be returned in a UI Toolkit application.

LPQUE

The FORM and ALIGN qualifiers do not affect printing on Windows unless system option #22 is set. The LPNUM value must be an alpha variable specifying the printer device in the form "*name*, [*device*]", unless system option #22 is set.

OPEN

The following OPEN statement qualifiers are ignored (these are only meaningful on OpenVMS):

- BKTSIZ
- BLKSIZ
- BUFNUM
- BUFSIZ
- CONTIG
- DEQ
- RECTYPE

OPTIONS qualifier

The following options for the OPTIONS qualifier are ignored (these are only meaningful on OpenVMS):

- /bufnum
- /bufsiz
- /deq
- /rectype
- /stream

READS

A difference of behavior exists between Windows and other platforms when performing a READS from the terminal. On UNIX and OpenVMS, any text that was previously displayed at the location of the READS remains displayed until the user types over it. On Windows, this text is erased for the width of the buffer passed to READS. The reason for this difference is that on Windows, the input is performed using an edit control to give the user thebenefit of editing features (arrow keys, home/end, cut/copy/paste, and so forth.). In order for text to be displayed within the edit control, it would also have to be returned to the program if the user merely pressed ENTER. This would not be operationally equivalent to the behavior on UNIX and OpenVMS systems, where the text, although displayed, is not returned in the READS buffer unless the user types it in. We therefore opted for keystroke compatibility, as opposed to visual compatibility, with other systems. The behavior on UNIX and OpenVMS is actually the less consistent of the two, but because it has historical precedent, we cannot change it without breaking existing code.

If you want to be able to prefill an area for input, use a different input method. Possibilities include various UI Toolkit routines (I_INPUT, I_INPFLD, U_FLD) or the low-level windowing routine W_DISP with the WD_READS subfunction.

Synergy Language functions

%DLL_*xxx* functions

The %DLL_*xxx* functions allow you to call subroutines in Windows 32-bit Dynamic Link Libraries (DLLs) and UNIX shared libraries (**.so** files). See the "Synergy DLL API" chapter of the *Synergy Language Reference Manual* for details and function syntax.

$SCR_ATT

The $SCR_ATT function sets screen attributes. The attributes BOLD, UNDERLINE, REVERSE, and NORMAL are supported. The BLINK attribute (to produce blinking text) is not supported in Windows. If set, BLINK will produce italicized text.

Italicized text extends beyond the bounds of a character cell; therefore, part of the last italicized character (in the window or preceding nonitalicized text) will not be visible.

Synergy Language subroutines

Using the following subroutines may make your programs nonportable. Synergy/DE on Windows ignores any system-supplied subroutines that are not appropriate for the Windows environment.

Subroutines that work differently on Windows

ENVRN

The ENVRN subroutine supports the following Windows operating system codes:

- For Windows XP, *system* is 18 and *machine* is 101 or 104.
- For Windows Server 2003, *system* is 19 and *machine* is 101 or 104.
- For Windows Vista, *system* is 20 and *machine* is 101 or 104.
- For Windows Server 2008, *system* is 21 and *machine* is 101 or 104.
- For Windows 7, *system* is 22 and *machine* is 101 or 104.
- For Windows Server 2008 R2, *system* is 23 and *machine* is 101 or 104.

JBNO

The JBNO routine returns the current process ID of the application as “ID.” JBNO returns the desktop’s window handle as the “parent ID.” JBNO returns the network adaptor card ID number as the “group ID.” For more information, see the JBNO routine in the “System-Supplied Subroutines and Functions” chapter of the *Synergy Language Reference Manual*.

KILL

The KILL subroutine terminates the current process but doesn’t log out as on multi-user systems.

RENAM

Because of physical limitations, the RENAM subroutine cannot rename files across logical drives. For example, the following statement is *not* allowed:

```
xcall renam("c:\target.ddf", "d:\source.ddf")
```

In this situation, a copy must be done, which is beyond the scope of RENAM.

RUNJB

The RUNJB subroutine supports only two arguments: PROGRAM and PID. The definitions remain the same as on UNIX, but the following arguments to RUNJB will be ignored when running on Windows:

- TERMINAL
- SUBPROCESS
- IO_FLAG

The maximum number of processes is limited only by system resources.

SETLOG

The SETLOG subroutine affects only the current environment and child processes. When a program terminates, the specified environment variable is the same as when the program began.

SHELL

The SHELL subroutine has a third argument specifically for Windows. For more information see SHELL in the "System-Supplied Subroutines and Functions" chapter of the *Synergy Language Reference Manual*.

SPAWN

The SPAWN subroutine's second argument has specific options for Windows. See SPAWN in the "System-Supplied Subroutines and Functions" chapter of the *Synergy Language Reference Manual*.

TNMBR

The TNMBR subroutine uses the TNMBR environment variable setting for the system terminal number.

Synergy Language subroutines included for compatibility

The GTPPN subroutine returns 0 in Windows environments and is provided for compatibility.

The PAINT subroutine is ignored on Windows.

The following UNIX subroutines are ignored on Windows:

- EXEC
- TFLSH

Record locking

If you're migrating from UNIX, be aware that record locks on Windows are channel-based. If the same program opens the same file on two different channels in update mode, both channels will be affected by each other's locks, which may cause unexpected $ERR_LOCKED errors.

Windowing subroutines

This section describes the Synergy Language windowing subroutines and functions that work differently in Windows. If you are using the Synergy Language windowing routines, you should review this section for highlights on the subroutines and functions that either work differently or are not supported.

Windows-specific subroutines

The W_CAPTION subroutine allows you to retrieve or load the window caption. This subroutine is specific to Windows environments and will be ignored on UNIX and OpenVMS. For more information see W_CAPTION in the "Synergy Windowing API" chapter of the *Synergy Language Reference Manual*.

Routines that work differently on Windows

W_AREA

When using the WA_ATTR function, ATTR_BLINK will display italic typeface.

Italicized text extends beyond the bounds of a character cell; therefore, part of the last italicized character (in the window or preceding nonitalicized text) will not be visible.

W_BRDR

On Windows, Synergy/DE supports three border options: no border, dialog frame, and caption frame. Specifying no border on Windows is the same as specifying no border on UNIX or OpenVMS. The dialog frame border contains only a solid frame around the window. The caption frame border contains the system menu box and the drag bar (with space for the title, or caption). Users can drag windows with a caption frame border. Caption frame is the default border type (see WB_DRAGON below), and only caption frame windows can be moved.

Keep in mind that window borders do not take up an entire "cell" as they do on UNIX and OpenVMS; if you are depending on one window's border to fully occlude a certain area of another window, this will not be the case. Because the border does not take up an entire cell, you may want to enable borders for Windows (but not for UNIX and OpenVMS) using the WB_NOCELL function.

Note the following W_BRDR functions:

- WB_OFF changes a window border to the no border style.
- WB_ON restores the window border back to its previous type (caption or dialog frame).
- WB_NOCELL behaves like WB_ON, but if the window is displayed on UNIX or OpenVMS, it will not have a border.
- WB_TITLE only works with caption frame borders, and the title is always positioned at WBT_TOP and WBT_CENTER.
- WB_DRAGOFF changes a caption frame border to a dialog frame border. If the border is off as a result of WB_OFF, you won't see the border style change until you do WB_ON or WB_NOCELL.
- WB_DRAGON changes a dialog frame border to a caption frame border. This is the default window frame state, except for one-line windows, which default to WB_DRAGOFF.

The following W_BRDR functions are ignored in the Windows environment:

- WB_ATTR
- WB_CHAR
- WB_COLOR
- WB_PARTIAL
- WB_TATTR
- WB_TCOLOR
- WB_TPOS

W_DISP

Note the following W_DISP functions:

- WD_ACCEPT uses the caret to designate where text is to be entered.
- When using the WA_ATTR function, ATTR_BLINK will display italic typeface. See the note on page 2-6.
- WD_GETS is implemented exactly like WD_READS (see WD_READS below).
- WD_READS uses a native Windows edit control sized according to the size of the field being read into. When the cursor is moved over the edit control, it changes to an I-beam and, when you click the mouse, places the caret within the field. Double-clicking the mouse highlights the field contents. You can also click and drag to highlight a portion of the field. When any or all of the field is highlighted, typing a character will replace the highlighted text. WD_READS

will terminate on ENTER, EOF, or any extended key press (function key or up and down arrow keys). Left and right arrow keys are processed by the edit control to move within the field, rather than being returned to the caller. Three routines elicit the terminate character:

%RDTRM

RDTRM

%RTERM

W_FLDS

Note the following W_FLDS functions:

- Input done through WF_INPUT will override the WF_ATTR and WF_COLOR subroutines since WF_INPUT uses a native edit control.
- Input done through WF_INPUT will not return extended keys in the normal manner. Extended keys will be accelerated if there is an entry in the Windows accelerator table, and a value other than the scan code will be returned.
- WF_INPUT is supported only through UI Toolkit. We do not recommend direct use of WF_INPUT.

W_INFO

The following W_INFO functions are ignored in the Windows environment:

- WI_BCHR
- WI_XFR (WIX_SAGET)
- WI_XFR (WIX_SDGET)

The WI_WINDOW function always returns a value of 0 for the occlude flag.

%W_INFO

The following %W_INFO function is ignored in the Windows environment:

- WIF_OCLFLG

W_INIT

The *#rows* and *#cols* arguments specify the initial size of the window only. If they are not passed, the APP_WIDTH and APP_HEIGHT initialization file settings will determine the initial size of the window. If these settings are not specified, the screen size will be 80 x 25.

W_PROC

Note the following W_PROC functions:

- WP_CURSOR will affect only the cursor used with WD_ACCEPT.
- The foreground and background numbers for WP_PALET can be between 0 and 511 on Windows but only between 0 and 255 on UNIX and OpenVMS; the significance of the values is system specific. (On Windows, these values are Synergy Language colors, which can be defaults set by the Synergy runtime or overrides set by the COLOR*n* environment variables or previous calls to W_PROC.)
- Using WP_RESIZE to switch to 132 columns also requires setting FONT_ALTERNATE in **synergy.ini** or **synuser.ini** to resize the characters.

The following W_PROC function is ignored in the Windows environment:

- WP_OPTION

UI Toolkit on Windows

This section provides general information only. Specific syntax and usage rules can be found in the *UI Toolkit Reference Manual*.

Script commands

The following window script commands and qualifiers function differently on Windows than they do on other operating systems. Those that are only applicable to Windows are marked as "Windows only." The "Windows only" commands are ignored on other operating systems but are retained in the window library, so that the same window library can be used in both Windows and UNIX or OpenVMS (little-endian) environments.

.BORDER

Renditions specified in conjunction with .BORDER have no meaning in a Windows environment and are ignored. Use the Windows Control Panel to change border renditions.

.BORDER DRAGBAR

(Windows only) The DRAGBAR qualifier to the .BORDER command creates a window that has a border and a dragbar. The dragbar is the portion of a window's border in which the caption is displayed. If the user clicks on the dragbar and drags the mouse, the window moves with it. For windows created at runtime, use the WB_DRAGON or WB_DRAGOFF options to the W_BRDR subroutine to create or remove dragbars.

By default, one-line input windows do not have a dragbar.

.BORDER NOCELL

(Windows only) The NOCELL qualifier to the .BORDER command designates that a window should only have a border if the border does not require a full character cell for display. In other words, the window has a border on Windows but not on UNIX and OpenVMS. For windows created at runtime, you can pass the WB_NOCELL option to the W_BRDR subroutine.

.COLUMN RIGHT and CENTERED

The *justification* argument for .COLUMN is ignored on Windows

.ENTRY NORESET

The NORESET qualifier to the .ENTRY script command is ignored.

.ENTRY RIGHT and CENTERED

The RIGHT and CENTERED qualifiers to the .ENTRY script command do not generate expected results. Menu entry text should always be left-justified (the default).

.FIELD NOTERM

The NOTERM qualifier to the .FIELD script command is ignored.

.FIELD PAINT

Paint characters specified by the PAINT qualifier to the .FIELD script command are ignored.

.ITEM RIGHT and CENTERED

The RIGHT and CENTERED qualifiers to the .ITEM script command do not generate expected results. Selection items should always be left-justified (the default).

.ITEM SELECT

Quick-select characters specified by the .ITEM script command are ignored. The first non-blank character of the entry is always used as the quick-select character. See also S_SELBLD on page 2-13.

.PAINT

Paint characters specified by the .PAINT script command are ignored.

.TITLE

Window titles are always in the top border, and they are always left-justified (indicated as BEGINNING) regardless of their settings in the script file. As a result, you can maintain them differently on UNIX and OpenVMS.

The options for the *rendition* argument have no meaning in a Windows environment and are ignored. Use the Windows Control Panel to change title renditions.

UI Toolkit subroutines

The following routines function differently on Windows than they do on other operating systems. Those that are only available on Windows are marked as “Windows only” and are ignored on other systems.

E_SECT (D_HEADER and D_CAPTION options)

If the size of your application header is set to 1 or 0 or is not specified, the header text (caption) passed to E_SECT appears in the title bar. You can use D_HEADER or D_CAPTION to set or modify the header text. For compatibility between all environments, we recommend that you always use D_HEADER.

If the size of your application header is greater than 1, an area below the title bar is used for displaying the header. Use D_HEADER to access this area. Use D_CAPTION to set and modify an additional line of text in the application's title bar.

The maximum visible length of header text (caption) is 78 characters.

Specifying D_CENTER or D_RIGHT for the header text (caption) is ignored. Captions on Windows are always left-justified.

EFKEY_METHOD

The user-overloadable EFKEY_METHOD is not supported on Windows.

L_INPUT

The *no_termination* argument is ignored on Windows. It is always treated as D_NOTERM.

L_SECT (D_TITLE)

Specifying D_CENTER or D_RIGHT for the title text is ignored. Titles on Windows lists are always left-justified.

L_SECTDRAW

On Windows, L_SECTDRAW either adds unexpected characters to the header or footer or is ignored. For standard (non-ActiveX) lists, L_SECTDRAW may add unexpected characters if it is followed by an L_SECT call that doesn't erase existing text. L_SECTDRAW is ignored for ActiveX Toolkit lists.

LLOAD_METHOD

The use of a list load method is required when doing list processing on Windows.

M_PROCESS

Typically, Windows applications do not pull menu columns down automatically. Instead, menus are explicitly invoked by the user using the mouse or the ALT key. Rather than change the default M_PROCESS behavior (which could require a redesign of your program flow), you may suppress the automatic menu pull-down feature of M_PROCESS by setting the environment variable DTK_MENU_UP or by calling M_DEFCOL(0).

For compatibility with other platforms, the *input_string* argument to M_PROCESS must match the keystrokes for a UNIX or OpenVMS environment, even though it requires fewer arrow movements to be pressed after the ALT key on Windows.

Additionally, *input_string* can only specify a submenu entry if it follows a valid menu entry. For example, the following is allowed:

```
xcall m_process("[menu_entry]<E>[submenu_entry]")
```

The following is not allowed:

```
xcall m_process("<R><D><D><E>[submenu_entry]")
```

S_SELBLD

Quick-select characters specified by S_SELBLD are ignored. The first nonblank character of the entry is always used as the quick-select character. See also .ITEM SELECT on page 2-11.

T_EDIT/T_VIEW

General windows that have been converted to text windows using T_SETUP display a scroll bar when necessary in T_EDIT but not in T_VIEW.

%TB_BUTTON

(Windows only) The %TB_BUTTON function enables you to load and manipulate toolbar buttons.

%TB_TOOLBAR

(Windows only) The %TB_TOOLBAR function enables you to create and manipulate an application toolbar.

U_ABORT

If **g_throwabort** is set to zero (the default), U_ABORT uses a Windows message box with a stop sign icon and an OK button.

U_ABOUT

U_ABOUT uses a Windows message box to display the “About” information. Text is left-justified, and the window contains an OK button and the application’s icon (the icon specified through U_ICON).

U_BAR

You cannot use U_BAR to remove the menu bar without first removing all menus.

U_CHARSB

The U_CHARSB subroutine does nothing in a Windows environment.

U_CREATESB

The U_CREATESB subroutine does nothing in a Windows environment.

U_DEBUG

The U_DEBUG subroutine brings up the Toolkit window debugger in a separate application window. The initial size and placement of this window is user-definable using the DBG_X, DBG_Y, DBG_WIDTH, and DBG_HEIGHT initialization settings in **synergy.ini**. The font is user-definable using the FONT_DEBUG setting.

U_EDITREND

Windows overrides many of the renditions set with U_EDITREND (as well as Proto and U_REND).

%U_ICON

(Windows only) The %U_ICON function defines the icon for a window, which is used when the window is minimized and represents the system menu.

U_MESSAGE

U_MESSAGE uses a Windows message box with an icon and an OK button. If you pass D_ERROR, the icon is an exclamation point (!). If you pass D_ALERT, the icon is an information icon (i).

%U_MSGBOX

%U_MSGBOX uses a Windows message box with an icon and one to three buttons, depending on the arguments passed. The icon can be a STOP sign, an exclamation point (!), a question mark (?), or the information (i) icon. Button choices are Yes, No, OK, Cancel, Abort, Retry, and Ignore.

U_POPUP

U_POPUP uses a Windows message box with an OK button.

%U_PRINTQUERY

(Windows only) The %U_PRINTQUERY function retrieves information about the currently selected printer or a specified printer from your Windows Print Manager. It can also retrieve the names of all configured printers.

%U_PRINTSETUP

(Windows only) The %U_PRINTSETUP function enables a user to change the default printer or printer properties.

U_REND

Windows overrides many of the renditions set with U_REND (as well as Proto and U_EDITREND).

U_START

Regardless of the number of lines specified for *footer_lines*, the maximum size of the footer is one line. If the *screen_rows* and *screen_columns* arguments are not passed, Toolkit uses the APP_WIDTH and APP_HEIGHT initialization settings in **synergy.ini** to determine the initial window size. If these are not set, the size will be 80 x 25.

U_UPDATESB

The U_UPDATESB subroutine does nothing in a Windows environment.

U_WAIT

U_WAIT uses a Windows message box with an OK button.

%U_WINHELP

(Windows only) The %U_WINHELP function invokes WinHelp Help. The API is nearly identical to the Windows SDK "WinHelp" function and enables the contents of the specified file to be displayed in the Help window.

%U_WNDEVENTS

(Windows only) The %U_WNDEVENTS function enables your application to respond to window mouse events. These events include mouse clicks, double-clicks, and moving, sizing, and closing a window.

%U_WNDFONT

(Windows only) The %U_WNDFONT function enables you to set or retreive font information. This function can set a specific font, or it can display a standard font dialog from which the user can select a font.

%U_WNDSTYLE

(Windows only) The %U_WNDSTYLE function enables you to change the vertical spacing on a window-by-window or list-by-list basis.

Other differences

The Toolkit debugger is displayed in a separate window. The debugger is invoked with XCALL U_DEBUG or by pressing CTRL+ R when the DTKDBG environment variable is set to 1.

Fill patterns do not extend into the area added to an input window for buttons; however, fill colors do.

Menu processing

Menu shortcut keys do not work when a menu is pulled down. We recommend that you set the environment variable DTK_MENU_UP or call M_DEFCOL(0).

Blank, line, and text menu entries can be highlighted but not selected.

Disabled menu entries can be highlighted but not selected.

Quick-select characters are always designated by an underscore.

The mouse or ALT and arrow keys can be used to process the menu. ALT+P, CTRL+P, or any other "process-menu" keys you may have defined do not always activate the menu.

Shortcut keys are mapped to a native Windows accelerator table to enable their use. The key mapping function available through Proto or U_EDITKEYS has an effect only on the shortcut text displayed in the menus. The actual keystrokes required for each function code are fixed. (For example, you cannot change the designated key from F4 to F2, but you can change "F4" to be represented as "PF4" when displayed in the menu.) The keystrokes required for each function code are reflected in the key map MSWINDOWS, which is the default key map for this environment.

Input processing

Input time-outs behave differently on Windows. On UNIX and OpenVMS, the time-out is reset after the user presses the first keystroke in an input field. On Windows, the time-out is not reset after the first or any other keystroke. Additionally, no time-out occurs while a menu is pulled down or when the application is otherwise suspended. When the application continues, the time-out occurs as soon as the application can process the message from the system.

List processing

When using UI Toolkit lists on Windows, the following limitations exist:

- Only single-line list entries are supported.
- Field-to-field mouse movement when performing input into a list is not supported.
- Disabled list entries are not supported.
- There is no visual difference between an enabled and disabled list; both use the same Windows Control Panel setting (Display > Item > Windows) for background and foreground colors.

Selection processing

Quick-select characters are supported but never highlighted, and invalid quick-select characters do not beep when pressed.

Text, line, and blank items have no meaning in a Windows environment. Selection list items should always be left-justified. Any other option does not operate as expected.

The following shortcut keys are captured by the selection window and are not returned to the calling application: the ARROW keys, the HOME key, and the END key. For this reason, and to reduce menu-bar flicker, the optional select menu passed to S_SELECT is not placed unless the global variable **g_plc_col_args** is true. See the *Migrating Your Application to Windows* document, available on the Online Manuals CD, for more information about **g_plc_col_args**.

Printing

As on other operating systems, the LPQUE statement is used to queue a file for printing by the system. This statement relies closely on the spooling facilities offered by the operating system.

You can set the PRINT_METHOD environment variable to send data from files printed with the LPQUE statement directly to the printer, without Print Manager affecting the escape codes.

System option #22 determines how the runtime handles the print job. If this option is set, the runtime sends LPQUE arguments to the file **DBLDIR:dblpq.bat**, which can contain customized print commands. If you do not set this option, the runtime sends the print job directly to the Microsoft Windows print manager.

Synergy Windows printing API

The Synergy Windows printing API enables you to use extended printer features in a true Windows environment to print documents and reports through Synergy Language functions. With these printing functions, you can embed graphics, preserve complex formatting, and perform print previews. See the *Migrating Your Application to Windows* document, available on the Online Manuals CD, for information and examples on how to implement this API in your applications.

Color

The WNDC environment variable is not used on Windows. Instead, a default color palette (with 16 default color palette entries) and a default set of Synergy color definitions (color 0 through color 511) are loaded into memory when the Synergy runtime starts. See "Colors and the color palette on Windows" in the "Synergy Windowing API" chapter of the *Synergy Language Reference Manual* for more information.

Part 2: Developing for UNIX

This section of the *Professional Series Portability Guide* contains information on Synergy Language that is specific to the UNIX operating system. When used in this document, the term “UNIX system” includes all machines on which UNIX, UNIX-derivative, or UNIX-compatible operating systems run.

3

Synergy/DE on UNIX: The Basics

System Requirements

Synergy Language will not run on UNIX unless you have Interprocess Communication (IPC) on your system. Most UNIX machines come with IPC, but check your system to make sure you have it. (IPC is a layered product on some machines.)

UNIX Characteristics

Case sensitivity

The UNIX file system is case sensitive, which means that it distinguishes lowercase letters from uppercase letters. As a result, Synergy Language on UNIX command lines are also case sensitive. If you enter a filename in uppercase letters and do not specify an extension, Synergy Language on UNIX will look for an uppercase extension. If you enter a filename in lowercase letters, Synergy Language on UNIX will look for a lowercase extension.

For example, if you enter

```
dbl DEMO
```

Synergy Language on UNIX will look for

```
DEMO.DBL
```

However, if you enter

```
dbl demo
```

Synergy Language on UNIX will look for

```
demo.dbl
```

Note that the OPEN statement on UNIX is also case sensitive; be careful to use the correct case for filenames in the OPEN statement.

Refer to DBLCASE in the "Environment Variables" chapter of *Environment Variables and System Options* for a way to override this behavior and externally enforce filename case from your program.

Big-endian and little-endian

Integers on UNIX systems are represented in either big-endian or little-endian format. For big endian, the low-order byte has the highest address. For little endian, the low-order byte has the lowest address. Computers with Motorola 68*xxx* series chips and non-Compaq RISC chips are big-endian systems. Computers with Intel and Compaq chips are little-endian systems.

Synergy Language executable programs and libraries can run on any system with the same endian format as the system on which they were built, with one exception: Version 8.1.5d or higher is required in order for object libraries to be portable between little-endian UNIX and Windows systems.

If your development system and the system you're porting to have the same endian type, you can use the same executables on both. If one is big endian and the other is little endian, you must re-create your executables on the system you're porting to.

Database files that contain integer data (including ISAM files) are not portable between big-endian and little-endian machines. You can use the %CNV_IP and %CNV_PI intrinsic functions to convert the integers in your database files to and from a portable format. Refer to the "System-Supplied Subroutines and Functions" chapter of the *Synergy Language Reference Manual* for details.

ISAM files that do not contain integer data (and may contain integer keys) are portable between big-endian and little-endian machines. Also, starting with version 7, integer data can be made portable with ISAM files. You can define where nonkey integer data resides within a record and have the conversion done automatically from portable to native form on read and write operations.

The following table lists the endian types of various systems.

Configuration	Endian type
HP OpenVMS Alpha	Alpha Native Little
HP OpenVMS I64	Itanium Native Little
HP Tru64 UNIX	64-bit Little
HP-UX 32-bit	Big
HP-UX 64-bit (PA-RISC)	64-bit Big
HP-UX 64-bit (Itanium)	64-bit Big
IBM AIX 5L 32-bit	Big
IBM AIX 5L 64-bit	64-bit Big
Linux 32-bit (Red Hat or SUSE)	Little
Linux 64-bit (x64)	64-bit Little
SCO OpenServer	Little
Sun Solaris 32-bit	Big
Sun Solaris 64-bit (x64/x86)	64-bit Little
Sun Solaris 64-bit (SPARC)	64-bit Big
Windows 32-bit	Little
Windows 64-bit (x86)	64-bit Little

Machine-specific characteristics

Linux

If you execute a script from the EXEC, SPAWN or RUNJB subroutines or the STOP or OPEN ("|") statements, Linux requires the first line to begin with "#! interpreter"; otherwise, the script will not execute. For example:

```
#! /bin/bash
```

If you are using option #22 for LPQUE, make sure the first line of **dblpq** includes this line.

SCO UNIX

There is a limitation with the **terminfo** entry for ANSI terminals on SCO UNIX systems. When "attributes-off" is set, graphics are turned off as well. This causes problems in Synergy Language windowing subroutine output on ANSI terminals when a graphics character with an attribute set is immediately followed by a graphics character with no attributes set. In this case, the nonattribute characters use the standard character set instead of the graphics set. To work around this problem, change the ANSI definition of characteristic "sgr0" to "\E[0m". Note that the Synergy runtime uses the **terminfo** database by default. You can also rebuild the runtime to use the **termcap** database.

4

Statements and Subroutines on UNIX

Synergy Language Statements

ACCEPT

The ACCEPT statement and the WD_ACCEPT subfunction of the W_DISP subroutine map LF to CR LF when input is redirected. Most input files on UNIX are composed of lines containing characters followed by an LF. To properly handle this lack of a CR, one is added when ACCEPT or WD_ACCEPT encounters an LF.

DETACH

The DETACH statement is only available on UNIX.

LPQUE

The LPQUE statement, which causes a disk file to be queued for transfer to a spooled device, relies closely on the spooling facilities offered by each operating system. As a result, it works slightly differently and uses different options on different systems. On UNIX, LPQUE ignores the FORM and ALIGN options if system option #22 is not set. LPQUE uses the following UNIX commands, which don't have these options:

System V	`lp`
BSD	`lpr`

By setting system option #22, you can customize the **dblpq.bat** file to recognize all options. For more information, see "Printing" on page 6-2 of this manual and system option #22 in the "System Options" chapter of *Environment Variables and System Options*.

OPEN

On UNIX and OpenVMS systems, you can optionally execute a keyboard command through an opened pipe using a special syntax of the OPEN statement. Although the options below are not available on Windows and UNIX, they can be used on Windows or UNIX to access files remotely on an OpenVMS platform.

The following OPEN statement qualifiers are only available on OpenVMS:

- BKTSIZ
- BLKSIZ
- BUFNUM
- BUFSIZ
- CONTIG
- DEQ
- RECTYPE

The following options for the OPTIONS qualifier are only meaningful on OpenVMS:

- /bufnum
- /bufsiz
- /deq
- /rectype
- /stream

Refer to the documentation for the OPEN statement in the "Synergy Language Statements" chapter of the *Synergy Language Reference Manual* for more information.

SEND

On UNIX, the SEND statement has a maximum message length of 4080 bytes.

SLEEP

If system option #12 and the TBUF environment variable are set, the I/O buffer will be flushed before the SLEEP statement is executed.

Synergy Language Routines

The following routines are only available on UNIX:

- BREAK – Issue a break to the channel
- FORK – Split the current process
- STTY – Control terminal settings

The following routines function differently on UNIX than they do on other operating systems:

- For the GTPPN subroutine on UNIX, *project* is the group ID and *programmer* is the user ID. *Privilege* is 1 if you have root privileges.
- The JBNO subroutine handles the process group identification number differently on UNIX.
- The KILL subroutine on UNIX only kills those processes in the same process group ID as the processes that are executing the KILL. (The process group ID includes all jobs spawned by the same original log-in.)
- The SETLOG subroutine only affects the current environment and any child processes; when your program terminates, the specified environment variable is the same as when your program began. If you don't pass a translation value, the SETLOG subroutine will unset (or delete) the specified environment variable.
- The SHELL subroutine works differently on each operating system.
- The *mode* argument of the SPAWN subroutine has specific options on UNIX. On UNIX, the runtime resets terminal (tty) settings by default when SPAWN is executed.
- The TNMBR subroutine determines the terminal number differently on UNIX than it does on other operating systems. See page 5-2 for more information about terminal numbering.
- The *status* argument of the TTSTS subroutine can be returned with the number of pending characters on OpenVMS and some UNIX systems.
- The %DLL_*xxx* functions allow you to call subroutines in UNIX shared libraries (**.so** files).

Most of the subroutines mentioned above are described in the "System-Supplied Subroutines and Functions" chapter of the *Synergy Language Reference Manual*. The %DLL_*xxx* functions are described in the "Synergy DLL API" chapter of the *Synergy Language Reference Manual*.

Record Locking

Locks on UNIX are process-based; if the same program opens the same file on two different channels in update mode, one channel won't be affected by the other's locks. Also, if the same record is read by both channels, the record will be unlocked by one channel without the other channel being aware of it.

Setting the LOCK:Q_NO_LOCK qualifier will override an exclusively locked file that has been opened using the SHARE:Q_EXCL_RW qualifier, unless you enforce mandatory locking with protection at the operating system level (with **chmod**).

If a user opens a file on more than one channel, Synergy Language defers the actual system closing of the file until it has been closed on all channels opened by that user. This feature is necessary to maintain locks set by the other open channels. When the file is closed on one of the channels but remains open on other channels, Synergy Language "holds" the channel until the file has been closed on all of the channels.

If you're migrating from Windows or OpenVMS, be aware that record locks are no longer channel-based as they are on these systems.

5

Terminal Characteristics on UNIX

Terminal Numbers Used by Synergy Language

Synergy Language uses terminal numbers in the TNMBR subroutine and in one form of the SEND statement. Synergy Language determines the terminal number in one of three ways:

1. Synergy Language looks in the environment of the current process for the environment variable TNMBR = *number* and uses the specified number as the current terminal number. This method has one disadvantage: you can easily give the same terminal number to more than one terminal. For example, if two people define TNMBR as equal to 1 in their log-in file and both are logged in at the same time, both of their terminals will have the number 1.

2. Synergy Language looks in the file **/etc/ttys** for the terminal name of the current process and uses the line number (beginning with line number 0) in the file as the terminal number. For example, if your terminal is called **/dev/junk**, and your **/etc/ttys** file contains the lines

   ```
   console
   tty01
   tty02
   junk
   ```

   your terminal number will be 3 (because junk is on the fourth line, and the first line is 0). The advantage of this terminal numbering system is that it automatically assigns terminal numbers for the entire system, and it works for non-numeric terminal names.

On systems where the **/etc/ttys** file exists, you can edit the file and rearrange the lines to alter the terminal numbers; don't change the format of the lines. Other characters may appear on each line, but these characters are ignored; do not delete them. If the **/etc/ttys** file doesn't exist (which means your operating system doesn't use it), you can create your own **ttys** file. On systems with network devices, this file can be extremely large.

3. If you haven't set TNMBR and **/etc/ttys** doesn't exist, Synergy Language will assign a number based on the name of your terminal. This method works if all of your terminal names end in a unique number. However, if some of the numbers are duplicates or if the name is non-numeric, the results are unpredictable.

Terminal Settings Used by Synergy Language

By changing certain STTY settings, you can change default key functions like interrupt and backspace. For example, UNIX commonly uses the delete key as interrupt and the backspace key as erase by default, but you might want to change these default settings to the Compaq style, making the delete key erase and CTRL+C interrupt. To do so, follow these steps:

1. View all of your terminal settings, as follows:

System V	`stty -a`
BSD	`stty everything`

2. Change your settings from the command line:

   ```
   stty intr  CTRL+C
   stty erase  DELETE
   ```

   You must actually press the DELETE and CTRL+C keys (and then press ENTER) to change the settings.

3. To add these settings to your **.profile** or **.login** file, type the following lines in your file:

   ```
   stty intr "^c"
   stty erase "^?"
   ```

   At runtime, Synergy Language makes the following changes to the terminal settings:

   - Turns ECHO off
   - Turns ICANON off
   - Turns ICRNL off
   - Sets VMIN to 1

   All other settings are left alone.

   See your UNIX command manual for more information about changing terminal settings.

   You can also control terminal settings using the STTY subroutine. See STTY in the "System-Supplied Subroutines and Functions" chapter of the *Synergy Language Reference Manual* for information.

Synergy Language and the UNIX Terminal Database

The escape sequences used by VT100-class ANSI terminals are known by the Synergy runtime on UNIX. If you are using a VT100 terminal and have set the TERM environment variable to **vt100**, *you don't need to read the rest of this section*; everything used by the Synergy runtime is already available.

System option #30 sets VT102 escape sequences. See system option #30 in the "System Options" chapter of *Environment Variables and System Options* for more information.

Synergy Language on UNIX uses one of the following terminal database files for non-VT100 screen displays: **/etc/termcap** or **/usr/lib/terminfo/***. If you have a non-VT100 terminal, you may need to modify the entry for your terminal.

The difference between **termcap** and **terminfo** is that **termcap** is a general purpose database that allows direct access to specific terminal capabilities, while **terminfo** is a compiled database that requires specific routines to access terminal entries. Because **terminfo** has a defined structure of terminal capabilities that it recognizes, there's no way to add new unsupported capabilities.

If your operating system normally has **terminfo**, the Synergy runtime will be built with the **terminfo** database. You can verify that your runtime is built with **terminfo** by entering the **what** system command:

```
what $DBLDIR/bin/dbr
```

(This command may be a layered product on your system and may not be available.)

Among other things, the **what** utility reports which terminal database is being used. A runtime file, **dbr.tc**, is commonly distributed that includes the **termcap** library support.

If you want to rebuild the runtime to include some C interface routines, you'll need to reconfigure your runtime by changing the terminal library in the **$DBLDIR/csrc/makedbr** file. For instance, to use **termcap** instead of **terminfo**, do the following:

1. Edit the **makedbr** file to assign the **termcap** library to the variable TLIB. For example:

   ```
   TLIB="$DBLDIR/lib/tclib.a"
   ```

2. Run **makedbr** to create the new runtime. For example:

   ```
   makedbr
   ```

Determining which database you should use

To determine which terminal database you should use, first find out if your terminal emulates a VT100-class ANSI terminal. If it does, set your terminal up as a VT100-class ANSI terminal and set TERM to vt100.

If you're using a terminal that cannot emulate a VT100-class ANSI terminal, and your application requires some capability that your terminal isn't performing properly, you'll need to modify the terminal database. If the runtime is built with **terminfo**, modify the **terminfo** data file for your terminal by doing the following:

1. Look for the **terminfo** source file for your terminal. This file should have the extension **.src**. If your system distributes this source file, you may find it in /usr/lib/terminfo or one of its subdirectories.
2. If you can't find a distributed **terminfo** source file, you may need to create one. You can either create the source file using **infocmp**, which is available with System V Release 3 (but not with any previous release), or you can create it from scratch.
3. If the **terminfo** file isn't available for your terminal, you will need to create one.
4. After making any of the above changes, compile the **terminfo** source file using the **tic** compiler (if it's available on your system).
5. Check your changes by running the **tstat.ti** utility. See "The terminal capabilities status program" on page 5-7 for more information.

If you need to add a capability that isn't available with **terminfo**, or if you can't complete any one of the steps listed above, you'll have to use **termcap** instead.

If you're familiar with **terminfo**, go ahead and use it; but we recommend that you use **termcap** for the following reasons:

- **Termcap** is not compiled, which makes it easier to change.
- Not all Synergy Language features are supported by **terminfo**.

Terminal database file syntax

The syntax of a **termcap** entry is as follows:

names:*code=esc_seq*:*[code=esc_seq*:*…]*

The syntax of a **terminfo** entry is as follows:

names, *code=esc_seq*, *[code=esc_seq*:*…]*,

Arguments

names

Lists the three names that are known for the terminal. The names are separated by the vertical bar character (|). The first name is always two characters long for compatibility with older systems. The second name is the most common abbreviation for the terminal and the name used in the TERM environment variable setting. The third name should be a long name that fully identifies the terminal. Only the third name can contain blanks for readability.

code

The terminal capability code. All of the terminal capability codes used by Synergy Language are listed in "Terminal codes used by Synergy Language" on page 5-13.

esc_seq

The escape sequence (or substitute character, in the case of a graphics character) for the terminal capability code. Look for the escape sequences in your system or terminal documentation.

Discussion

A few special codes that are significant to Synergy Language do not use the *code=esc_seq* syntax:

- bs
- ms
- sg#1

You can list as many *code=esc_seq* groups as you want, as long as **termcap** groups are separated by colons (:) and **terminfo** groups are separated by commas (,).

Termcap entries must be one continuous logical line. The continuation character for a **termcap** entry is a backslash (\). **Terminfo** entries, on the other hand, don't require a single line, so no continuation character is necessary.

The following is the standard ANSI CRT **termcap** entry (\E represents an escape character):

```
li|ansi|Ansi standard crt:\
  :al=\E[L:ms:am:bs:cd=\E[J:ce=\E[K:cl=\E[2J\E[H:cm=\E[%i%d;%dH:co#80:\
  :dc=\E[P:dl=\E[M:do=\E[B:bt=\E[Z:ei=:ho=\E[H:ic=\E[@:im=:li#25:\
  :nd=\E[C:pt:so=\E[7m:se=\E[m:us=\E[4m:ue=\E[m:up=\E[A:\
  :kb=^h:ku=\E[A:kd=\E[B:kl=\E[D:kr=\E[C:eo:\
  :sf=\E[S:sr=\E[T:\
  :cc=\E[2K:cb=\E[1K:CT=\E[1J:\
  :md=\E[1m:mb=\E[5m:mr=\E[7m:me=\E[m:\
  :ac=x\263u\264k\277m\300v\301w\302t\303q\304n\305j\331l\332:\
  :CW=\E[M:NU=\E[N:RF=\E[O:RC=\E[P:\
  :WL=\E[S:WR=\E[T:CL=\E[U:CR=\E[V:\
  :HM=\E[H:EN=\E[F:PU=\E[I:PD=\E[G:\
```

```
  :fD=\E[30m:fB=\E[34m:fG=\E[32m:fC=\E[36m:\
  :fR=\E[31m:fM=\E[35m:fY=\E[33m:fW=\E[37m:\
  :bD=\E[40m:bB=\E[44m:bG=\E[42m:bC=\E[46m:\
  :bR=\E[41m:bM=\E[45m:bY=\E[43m:bW=\E[47m:
```

The following is the standard ANSI CRT **terminfo** entry:

```
ansi|generic ansi standard terminal,
  am, xon, ms,
  cols#80, lines#24,
  bel=^G, blink=\E[5m, bold=\E[1m, cbt=\E[Z,
  clear=\E[H\E[J, cr=\r, cub=\E[%p1%dD, cub1=\b,
  cud=\E[%p1%dB, cud1=\n, cuf=\E[%p1%dC, cuf1=\E[C,
  cup=\E[%i%p1%d;%p2%dH, cuu=\E[%p1%dA, cuu1=\E[A,
  dch1=\E[P, dl=\E[%p1%dM, dl1=\E[M, ed=\E[J, el=\E[K,
  home=\E[H, ht=\t, hts=\EH,
  ind=\n, invis=\E[8m, kbs=\b, kcub1=\E[D, kcud1=\E[B,
  kcuf1=\E[C, kcuu1=\E[A, khome=\E[H,
  rev=\E[7m, rmso=\E[m,
  rmul=\E[m,
  sgr0=\E[0m, smso=\E[7m, smul=\E[4m, tbc=\E[2g,
```

When the Synergy runtime processes a screen function, it searches the database for the appropriate terminal capability entry. If the entry isn't found and there is no alternative capability, the runtime ignores the function and doesn't flag an error. If a capability doesn't work as you expect, you can either remove the entry from the database (if it isn't a crucial feature) or correct the entry. If your terminal has a capability that's not listed in the database, you can add the entry.

The terminal capabilities status program

We provide a utility called **tstat** (**tstat.tc** for **termcap** and **tstat.ti** for **terminfo**) that tests the terminal entry for your terminal type and identifies entries you may need to change.

Tstat has two test sections. The first section scans your terminal database file and lists terminal capabilities that are used by Synergy Language but that aren't defined in the database file. The second section is a series of tests for each Synergy Language screen function. As you run the tests, compare the results on the screen with the expected results. If the results are different, you may need to modify your terminal database file. Check your terminal documentation to verify that your terminal (defined by the TERM environment variable) can physically perform the screen function you need.

Tstat – first section

Set the environment variable TERM to specify your terminal type, and run the appropriate **tstat** program for your database (**tstat.tc** or **tstat.ti**). The first section of the program lists information for the following Synergy Language areas:

- TERM
- $SCR_POS
- $SCR_CLR
- $SCR_MOV
- $SCR_ATT
- Windows
- Cursor motion with attributes set

We'll use **tstat.tc** as an example. If **tstat.tc** finds the terminal entry in **/etc/termcap**, it lists the terminal type. If all of the required terminal capabilities exist in **termcap** for each of the above functions, **tstat** displays the word "Passed" next to the function. If **tstat** lists a capability as missing, the **termcap** entry for your terminal type doesn't define it. (However, your terminal may still be able to perform that function.)

The Windows section of the test checks for "line wrap off" (RA) and "line wrap on" (SA). In some cases, a window that is placed on the right edge or bottom of the screen can cause the screen to scroll upward. If you can't change screen wrap through escape sequences in **termcap**, disable screen wrap at the hardware level if possible. If you can't disable screen wrap at all, position your windows away from the edges of the screen.

Your output from the first section of **tstat.tc** might look like this:

```
TERM=ansi
$SCR_POS:     Passed
$SCR_CLR:     Passed
$SCR_MOV:     Passed
$SCR_ATT:
  SAVE: cannot save attribute [sc]
  RESTORE: cannot restore attribute [rc]
Windows:      Passed

-- Cursor motion is ok with attributes set [ms present]
Press return to perform tests: [n to quit]
```

If you want to add any of the missing attributes to your **termcap** entry, type "n" to exit the program; otherwise, press ENTER to go on to the second section of **tstat**.

Tstat – second section

The second section of **tstat** performs six tests. For each test, press ENTER to run the test or type "n" to skip the test and go on to the next one. Only tests with existing terminal capabilities in the **termcap** file are run. The following prompts appear for each test.

Test 1: Press return to clear screen [n to skip]:

When you press ENTER, test 1 clears the screen and moves the cursor to the upper left corner. If anything else occurs, check the "clear screen" (cl) and "screen cursor positioning" (cm) entries in the **termcap** file.

Test 2: Press return to test screen positioning [n to skip, or #]:

Test 2 has an optional argument for the number of lines on your terminal. (The default is 23.) When you press ENTER, test 2 places a set of boxes on the screen. Each box is made of two square brackets. The right half of a box is in the upper left corner of the screen. The test also places an X in each box.

The box and X placement are designed to test the most critical (x,y) coordinates. If the Xs are not placed inside the boxes, either your screen doesn't have 23 lines (in which case you should rerun test 2 and specify the correct number of lines for your terminal at the prompt), or **termcap** has the wrong entry for "screen cursor positioning" (cm).

Test 3: Press return to test relative movement [n to skip]:

When you press ENTER, this test places a set of boxes on the screen and puts asterisks (*) inside them using relative screen movement. If the asterisks aren't inside the boxes, verify that "cursor up" (up), "cursor down" (do), "cursor right" (nd), and "cursor left" (bc) were found in the first section of the **tstat** utility; otherwise modify the **termcap** file to use the correct sequences.

Test 4: Press return to test line clearing [n to skip]:

When you press ENTER, this test places four rows of Xs on the screen. It then tests the following clearing functions: EOL, EOS, BOL, BOS, and LINE. The four rows of Xs are redrawn after each clear function. If any of the functions don't clear the correct part of the screen, check the following **termcap** entries: "clear to end of line" (ce), "clear to end of screen" (cd), "clear to beginning of line" (cb), "clear to top of screen" (CT), and "clear line" (cc).

Test 5: Press return to test attributes [n to skip]:

When you press ENTER, this test lists the name of each screen attribute and several groups of attributes on separate lines, then sets the appropriate attribute for each line. Note that when several attributes are set in one statement, Synergy Language doesn't combine them but sets them one at a time. For more information on screen attributes, See "Screen attributes" on page 5-13.

Test 6: Press return to graphics [n to skip]:

When you press ENTER, test 6 displays a graphics character list, which is required by the windowing subroutines.

This test displays the graphics character after each description in square brackets. If any of the characters don't match their description, check the appropriate **termcap** entry. Turn a graphics character set on and off with the **termcap** entries "graphics start" (GS) and "graphics end" (GE). See "Screen graphics" on page 5-15 for more information.

Sample tstat.tc session

The following example leads you through a sample **tstat.tc** test and revision session for an ANSI terminal. We'll assume the original **/etc/termcap** file contains this entry:

```
li|ansi|Ansi standard crt:\
:al=\E[L:am:bs:cd=\E[J:ce=\E[K:cl=\E[2J\E[H:\
:cm=\E[%i%d;%dH:co#80:dc=\E[P:dl=\E[M:do=\E[B:\
:bt=\E[Z:ei=:ho=\E[H:ic=\E[@:im=:li#25:\
:nd=\E[C:ms:pt:so=\E[7m:se=\E[m:us=\E[4m:ue=\E[m:up=\E[A:\
:kb=^h:ku=\E[A:kd=\E[B:kl=\E[D:kr=\E[C:eo:\
:sf=\E[S:sr=\E[T:\
:GS=\E[12m:GE=\E[10m:GV=\63:GH=D:\
:GC=b:GL=v:GR=t:\
:G1=?:G2=Z:G3=@:G4=Y:\
:GU=A:GD=B:RT=^J:\
:CW=\E[M:NU=\E[N:RF=\E[O:RC=\E[P:\
:WL=\E[S:WR=\E[T:CL=\E[U:CR=\E[V:\
:HM=\E[H:EN=\E[F:PU=\E[I:PD=\E[G:
```

With the above **termcap** entry, **tstat.tc** gives the following information in its first section:

```
TERM = ansi
$SCR_POS: Passed
$SCR_CLR:
  LINE: missing clear line capability [cc]
  BOL: missing clear to beginning of line capability [cb]
  BOS: missing clear to top of screen capability [CT]
$SCR_MOV: Passed
$SCR_ATT:
  SAVE: cannot save attribute [sc]
  RESTORE: cannot restore attribute [rc]
Windows:
  Missing cursor off capability [CF]
  Missing wrap off capability [RA]

-- Cursor motion is ok with attributes set [ms present]
Press return to perform tests: [n to quit]
```

Tstat.tc shows that **termcap** needs the following entries: cc, cb, CT, sc, rc, CF, and RA. We'll type **n** to stop the program and edit the **termcap** file. Let's assume our example program doesn't save or restore attributes and doesn't need the missing window capabilities.

We'll add the following line to the **termcap** file:

```
:cc=\E[2K:cb=\E[1K:CT=\E[1J:\
```

Now we'll run **tstat.tc** again. The results should look like this:

```
TERM = ansi
$SCR_POS: Passed
$SCR_CLR: Passed
$SCR_MOV: Passed
$SCR_ATT:
  SAVE: cannot save attribute [sc]
  RESTORE: cannot restore attribute [rc]
Windows:
  Missing cursor off capability [CF]
  Missing wrap off capability [RA]
-- Cursor motion is ok with attributes set [ms present]
Press return to perform tests: [n to quit]
```

These results work with our imaginary Synergy Language application. We'll press ENTER and begin the next section of **tstat.tc**. Our first prompt looks like this:

Test 1: Press return to clear screen [n to skip]:

When we press ENTER, the screen clears and the cursor moves to the top left corner, which means that the "clear screen" (cl) entry is correct. The second prompt is as follows:

Test 2: Press return to test screen positioning [n to skip, or #]:

We'll press ENTER to use the default number of screen lines (23). The screen clears, and seven and one-half boxes appear on the screen. An X is placed *above* each box, which means that our terminal has more than 23 screen lines. We'll press the interrupt character and rerun test 2, this time specifying the correct screen length instead of just pressing ENTER. Now the Xs are correctly placed inside each box, and we know "screen cursor positioning" (cm) is correct. The third prompt is as follows:

Test 3: Press return to test relative movement [n to skip]:

When we press ENTER, the screen clears and then four boxes appear. An asterisk fills each square, indicating that the "cursor up" (up), "cursor down" (do), "cursor right" (nd), and "cursor left" (bc) entries work. The fourth prompt is as follows:

Test 4: Press return to test line clearing [n to skip]:

When we press ENTER, the test displays four rows of Xs with the screen attribute EOL in the center. The test clears to the end of the line, redraws the four rows of Xs, and tests the EOS, BOL, BOS, and LINE attributes. Each works correctly. The fifth prompt is as follows:

Test 5: Press return to test attributes [n to skip]:

After we press ENTER, the following words are listed in one column on the left side of the screen: BLINK, BOLD, UNDERLINE, REVERSE, BOLD + UNDERLINE, and BLINK + BOLD + REVERSE + UNDERLINE. Each word (or group of words) should be displayed in its screen attribute, but BLINK, BOLD, and REVERSE appear in reverse video.

The following attributes aren't tested because they aren't defined in **termcap**: SAVE GRAPHICS, RESTORE GRAPHICS, and CURSOR OFF. (If your terminal supports these attributes and you define them in **termcap**, they will also be tested.) The final prompt is as follows:

Test 6: Press return to graphics [n to skip]:

We'll press ENTER to continue testing. The list of graphics characters is displayed, but the last three characters are wrong. Therefore, we'll need to redefine the **termcap** entries "left tee" (GL), "right tee" (GR) and "center crossing" (GC).

One complete **tstat.tc** round is now completed. Now we'll add the "blink," "bold," and "reverse" entries and fix the graphics characters in the **termcap** file. We'll look in our terminal manual for the correct entries and make the following changes to **/etc/termcap**:

We'll change

```
:GC=b:GL=v:GR=t:\
```

to

```
:GC=E:GL=C:GR=\64:\
```

and add

```
:md=\E[1m:mb=\E[5m:mr=\E[7m:\
```

Now we'll rerun **tstat.tc** to check how our changes affect tests 5 and 6. If each test works, testing is finished. The final **/etc/termcap** entry should look like this (changed lines are marked with an asterisk):

```
li|ansi|Ansi standard crt:\
:al=\E[L:am:bs:cd=\E[J:ce=\E[K:cl=\E[2J\E[H:\
:cm=\E[%i%d;%dH:co#80:dc=\E[P:dl=\E[M:do=\E[B:\
:bt=\E[Z:ei=:ho=\E[H:ic=\E[@:im=:li#25:\
:nd=\E[C:ms:pt:so=\E[7m:se=\E[m:us=\E[4m:ue=\E[m:up=\E[A:\
:kb=^h:ku=\E[A:kd=\E[B:kl=\E[D:kr=\E[C:eo:\
:sf=\E[S:sr=\E[T:\
      *
:cc=\E[2K:cb=\E[1K:CT=\E[1J:\
:GS=\E[12m:GE=\E[10m:GV=\63:GH=D:\
      *
:GC=E:GL=C:GR=\64:\
:G1=?:G2=Z:G3=@:G4=Y:\
:GU=A:GD=B:RT=^J:\
      *
```

```
:md=\E[1m:mb=\E[5m:mr=\E[7m:\
:CW=\E[M:NU=\E[N:RF=\E[O:RC=\E[P:\
:WL=\E[S:WR=\E[T:CL=\E[U:CR=\E[V:\
:HM=\E[H:EN=\E[F:PU=\E[I:PD=\E[G:
```

Terminal codes used by Synergy Language

Below is a list of functions used by Synergy Language and the **termcap** and **terminfo** codes that affect them. Some codes used in **termcap** have alternates. Some alternates have the same functionality as their main **termcap** requirement but different names. Other alternates produce slightly different results and may not work as expected. The alternate for "cursor up" (ku), for example, may function differently than its main requirement. Some keyboards generate the same escape sequence for the keyboard up arrow key as the terminal requires for the cursor up function, others do not.

We have identified a problem with the **terminfo** entry for ANSI terminals on SCO UNIX systems. When "Clear attributes" (sgr0) is set, graphics are turned off as well. This situation causes problems in Synergy Language windowing subroutine output on ANSI terminals when a graphics character with an attribute set is immediately followed by a graphics character with no attributes set. In this case, the nonattribute characters use the standard character set instead of the graphics set. To work around this problem, you can change the ANSI definition of characteristic sgr0 to \E[0m. Another alternative is to rebuild the runtime to use the **termcap** database.

Screen attributes

Termcap traditionally defines only "standout" (so) and "underline" (us) capabilities for screen attributes. We've added entries for "bold" (md), "blink" (mb), and "reverse" (mr) to increase screen attribute portability. These codes came from UNIX System V **terminfo** definitions.

If you terminal supports these attributes, you can add the md, mb, and mr entries to your **termcap** file. If the runtime doesn't find these entries, it will use "standout" (so), which is usually reverse video.

The same situation exists for clear attributes. "Standout end" (se) and "underline end" (ue) are normally the only attribute clearing entries in **termcap**, so we added the entry "clear attributes" (me) to clear all attributes. If your terminal supports a single escape sequence to clear attributes, we recommend defining **me** to that sequence; otherwise, define **me** as all of the clearing sequences in one string.

For example, if your terminal has the clear sequences ESC[1n (clear bold), ESC[2n (clear underline), ESC[3n (clear reverse), and ESC[4n (clear blink), the following **termcap** definition clears all attributes at once in a Synergy program:

```
me=\E[1n\E[2n\E[3n\E[4n
```

If the screen attributes extend to the end of each line, delete the “move in standout” mode (ms) entry from your **termcap** file. The Synergy Language windowing subroutines won’t work correctly on terminals that don’t support “move in standout” mode if the ms code is specified. If the attributes appear to be normal, be sure that ms is present, because it increases screen performance.

Be sure the lines begin at the far left side of the screen. If they start one column to the right, your terminal uses embedded attributes. (In other words, setting or clearing an attribute uses a character position on the screen.) Unfortunately, this means that Synergy Language cannot support attributes on your terminal. To make Synergy Language ignore attribute calls, be sure to add the following special **termcap** entry for embedded attributes:

```
sg#1
```

Screen graphics

Character description	Character	Termcap entry	"ac" index
Upper-right corner	┐	G1	k
Upper-left corner	┌	G2	l
Lower-left corner	└	G3	m
Lower-right corner	┘	G4	j
Horizontal line	─	GH	q
Vertical line	│	GV	x
Bottom tee	┴	GU	v
Top tee	┬	GD	w
Left tee	├	GL	t
Right tee	┤	GR	u
Center crossing	┼	GC	n

If your terminal supports 8-bit graphics, you can use octal values for the graphics characters instead of ASCII characters. For example, if the upper-right corner character is octal 263, use the following **termcap** entry:

```
G1=\263
```

The character set definition is the only place 8-bit characters are recognized in the **termcap** file; other **termcap** functions strip off the eighth bit.

When changing a **termcap** entry, set the TERMCAP environment variable to a test file where you can modify the entry. (See TERMCAP in the "Environment Variables" chapter of *Environment Variables and System Options* for more information.) This will prevent you from accidentally disturbing other users' settings if you make an error.

Another, sometimes easier, method for setting graphics is the "ac" **termcap** entry. Here is a sample "ac" **termcap** entry:

```
:ac=x\263u\264k\277m\300v\301w\302t\303q\304n\305j\331l\332:\
```

The letters x, u, k, m, v, w, t, q, n, j, and l represent an index for the graphics characters as listed in the table above. (For example, x is the vertical line character, u is the right tee character, k is the upper-right corner character, and so forth.) The three-digit numbers preceded by a "\" are octal values that represent the graphics for your terminal.

Alternatively, instead of the *nnn* format, you can specify a single character that corresponds to the three-digit ASCII representation of the graphic for your terminal. For example, "xAuBkC" indicates that A equals the vertical line character, B equals the right tee character, and C equals the upper-right corner character.

The order of the index letters (x, u, k, and so on) in the "ac" **termcap** entry is not important, but each letter must immediately precede its *nnn* or *A* character.

Termcap code	Terminfo code	Function
[ac]	[acsc]	Graphics character list
[bs][bc][le]	[cub1]	Cursor left
[bx]	[box1]	Enable line drawing characters on the IBM RS-6000
[cb]	[el1]	Clear to beginning of line
<cc>		Clear current line
[cd]	[ed]	Clear to end of screen
[ce]	[el]	Clear to end of line
[CF][vi]	[civis]	Cursor off
[cl]	[clear]	Clear screen
[cm]	[cup]	Screen cursor positioning
[CO][ve]	[cvvis]	Cursor on
<CT>		Clear to top of screen
[do]	[cud1]	Cursor down
[eA]	[enacs]	Enable graphics character set
[ec]	[ech]	Erase character (high-speed clearing)
[G1]		Upper-right corner
[G2]		Upper-left corner

Termcap code	Terminfo code	Function
[G3]		Lower-left corner
[G4]		Lower-right corner
[GC]		Cross
[GD]		Top tee
[GE][ae]	[rmacs] (or [font0] on AIX)	Exit alternate character set
[GH]		Horizontal bar
[GL]		Left tee
[GR]		Right tee
[GS][as]	[smacs] (or [font1] on AIX)	Enter alternate character set
[GU]		Bottom tee
[GV]		Vertical bar
[mb][so]	[blink][smso]	Blink
[md][so]	[bold][smso]	Bold start
[me]	[sgr0]	Clear attributes
[mr][so]	[rev][smso]	Reverse start
[ms]	[msgr]	Move in standout
[nd]	[cuf1]	Cursor right
[RA]	[rmam]	Turn off line wrap
[rc]	[rc]	Restore cursor position and attributes
[SA]	[smam]	Turn on line wrap
[sc]	[sc]	Save cursor position and attributes
[sg]	[xmc]	Embedded attributes
[up]	[cuu1]	Cursor up
[us]	[smu1]	Underline start

The Synergy runtime internally defines the entries in angle brackets (< >). They are not standard **termcap** codes. Use the internal codes in your **termcap** file the same way you use the standard codes.

$SCR_ function	Windows function	Termcap code	Termcap alternates	Terminfo code
$SCR_POS(row,col)	WD_POS,row,col	[cm]		[cup]
$SCR_CLR(screen)		[cl][cm]		[clear][cup]
$SCR_CLR(eol)		[ce]		[el]
$SCR_CLR(eos)		[cd]		[ed]
$SCR_CLR(line)		<cc>		
$SCR_CLR(bol)		[cb]		[el1]
$SCR_CLR(bos)		<CT>		
$SCR_MOV(row,col)		[up] [do] [nd] [bc]	[ku] [kd] [kr] [kl][bs][le]	[cuu1] [cud1] [cuf1] [cub1]
$SCR_ATT(clear)	ATTR_CLR	[me]	[ue][se]	[sgr0]
$SCR_ATT(bold)	ATTR_BOLD	[md]	[so]	[bold][smso]
$SCR_ATT(under)	ATTR_UNDR	[us]		[smul]
$SCR_ATT(blink)	ATTR_BLNK or ATTR_ITAL	[mb]	[so]	[blink][smso]
$SCR_ATT(reverse)	ATTR_RVRS	[mr]	[so]	[rev][smso]
$SCR_ATT(save)		[sc]		
$SCR_ATT(restore)		[rc]		
$SCR_ATT(gon)		[GS]	[as]	
$SCR_ATT(goff)		[GE]	[ae]	

6

Other UNIX-Specific Information

Printing

You have two options when printing on UNIX:

- For nonspooled printers, you can open the device directly and write to it. For example:

  ```
  open(1, o, "/dev/tty02")
  ```

  However, if more than one person is printing at the same time, the print jobs will be intermixed.

- For spooled printers, use the LPQUE statement. For example:

  ```
  open(1, o, "printfile")
  .
  .
  .
  close 1
  lpque("printfile", lpnum:1, copies:2, delete)
  ```

DBLDIR:dblpq

DBLDIR:dblpq is a shell script file. You can customize printing by modifying this file to your own specifications. The runtime sends LPQUE arguments to this file if system option #22 is set. The LPQUE statement then executes the arguments in **dblpq** instead of those in the default printing program (which is **lp** on UNIX and **lpr** on 4.2BSD).

If you don't have a printer connected but you want to test your programs, you can set the environment variable PCMD as follows to cause LPQUE to send your output to your terminal:

```
pcmd=cat
```

If you do not specify an extension on a filename in your LPQUE statement, Synergy Language appends the default extension **.ddf**.

The LPNUM option

LPNUM is the LPQUE statement option that indicates to which spooled unit the file is to be directed. You can either specify the printer number or the printer name as the argument to LPNUM, although an alpha printer name specification is not portable to all operating systems. On a UNIX System V, specifying the number *n* sends output to a printer named "lp*n*." If you use a number on this system, you must run the UNIX **lpadmin** program and name your printers "lp1," "lp2," and so forth; **lpadmin** requires printers to be named, not numbered.

For example, the following statement sends a listing to a printer named “lp3”:

```
lpque("test.lis", lpnum:3)
```

while the statement below sends a listing to a printer named “laser”:

```
lpque("test.lis", lpnum:"laser")
```

On other systems, such as 4.2BSD, specifying the number *n* sends output to the printer numbered *n*. You can’t use alpha printer names on 4.2BSD because the printers are numbered.

Serial Ports

There are two ways to access a serial port on UNIX systems: uppercase letters (for example, **/dev/tty1A**) and lowercase letters (for example, **/dev/tty1a**). The uppercase form indicates use of CCITT modem control signal checking and timeouts.

To open a port using the uppercase form, use the /NODELAY qualifier for the OPTIONS qualifier for OPEN. If you do not specify this qualifier, the OPEN will fail after a time-out period. With Synergy Language 5.7.2 (or higher), if you specify the /NODELAY qualifier, you must reset the UNIX O_NDELAY system flag (which we set to allow the open) on the channel with the TTSTS subroutine. Doing so will enable subsequent ACCEPT/READS/GETS functions to operate normally. You must also use the INITPORT subroutine to set the optional *modem* argument to 1 (modem control, which is the default) so that the "hupcl" and "clocal" STTY settings are correctly set.

For correct operation of a serial line, including acceptance of all possible character codes, you must use INITPORT to set up the terminal characteristics. When using INITPORT, make sure it follows the OPEN statement *as soon as possible* in case data arrives on the serial port before INITPORT sets up the port speed in the UART. After a CLOSE, you must once again use INITPORT. (See INITPORT in the "System-Supplied Subroutines and Functions" chapter of the *Synergy Language Reference Manual* for more information.)

When using the lowercase form to access a serial port, set the INITPORT *modem* argument to 2 (no modem control). This setting requires "hupcl clocal" STTY settings to operate correctly.

Avoid using single-character GETS/ACCEPT sequences with TTSTS: this combination will slow down the system. A computer may be noticeably slowed by trying to accept characters at 9600 baud. Always use GETS with the WAIT and MASK qualifiers for timed I/O to a serial port. This allows several devices at 9600 baud to be handled simultaneously.

On some versions of UNIX, before the modem connection has been made (before there is a carrier), you must turn off **clocal** to issue modem commands. You can do this as follows:

- Set the mode argument of the INITPORT subroutine to 2.
- Issue the dial command.
- Set the modem control back to 1.

Windowing System

Enabling color

Enabling color on UNIX may involve an extra step that is not required by other operating systems. If TERM is not set to **xterm, ansi**, or **vt*xxx*** (for example, vt100 or any VT-series terminal setting), you must also add a set of codes to the **termcap** file entry for the terminal you intend to use. See WNDC in the "Environment Variables" chapter of *Environment Variables and System Options* for more information about these **termcap** codes.

Synergy does not support color from the **terminfo** database. If you would like to enable color in your application, you can do one of two things:

- If TERM is not set to **xterm**, **ansi**, or **vt*xxx*** but your terminal supports ANSI color, set the ANSICOLOR environment variable to use the built-in ANSI color sequences in the Synergy runtime to generate color. (See ANSICOLOR in the "Environment Variables" chapter of *Environment Variables and System Options*.)
- Use the **termcap** runtime, as described above.

(Note that if TERM is set to **xterm**, **ansi**, or **vt*xxx***, the runtime defaults to ANSI color escape sequences.)

Enabling hardware scrolling

The **termcap/terminfo** databases are accessed for the sequences that set scroll up, scroll down, and the scrolling region. If these sequences are not present, hardware scrolling will not be enabled. See "Synergy Language and the UNIX Terminal Database" on page 5-4 for more information.

System Options

The following options are either unique to UNIX or function differently in UNIX environments. Refer to the "System Options" chapter of *Environment Variables and System Options* for more information about each system option.

- System option #12, which determines whether or not you'll be able to customize the size of the terminal buffer, is only available on UNIX.
- System option #16, which maps the quit character to the interrupt character, is only available on UNIX and OpenVMS.
- If you set option #22, the runtime will send LPQUE arguments to the script file **DBLDIR:dblpq**, which you can change to your own printing specifications. The LPQUE statement will then execute the arguments in **dblpq** instead of those in the default printing program. See "Printing" on page 6-2 for more information on option #22 and LPQUE.
- System option #33 is only available on UNIX.
- A potential problem with setting system option #36, which enables the flushing feature of the runtime, is that some older UNIX operating systems may not support an **fsync()** routine. See system option #36 in the "System Options" chapter of *Environment Variables and System Options* for more information.

Message Facilities

To use the Synergy message manager on UNIX, you must set system option #7 with the DBLOPT environment variable. If you don't set option #7, Synergy Language will use the local message facility instead of the Synergy message manager for SEND and RECV statements.

Part 3: Developing for OpenVMS

This section of the *Professional Series Portability Guide* contains information about Synergy Language that is specific to the OpenVMS operating system. In this manual, the term "OpenVMS system" refers to any Alpha or I64 machine running the OpenVMS operating system.

7

Synergy/DE on OpenVMS: The Basics

OpenVMS Characteristics

Shared images

Synergy Language for OpenVMS uses a shared executable image for the runtime. Throughout this manual, when we refer to the runtime, we are referring to this shared executable image.

Some OpenVMS shared images on the Alpha are still translated images (EDTSHR, for example). Synergy Language applications cannot directly call these images unless XCALL linkage routines are compiled with the /tie option, which slows down every XCALL. To counteract this, we have implemented DBL$EXECUTE_IMAGE_ROUTINE, which activates the image "on the fly" in JACKET mode so that you can call these images without general performance degradation. If you use DBL$EXECUTE_IMAGE_ROUTINE and your call to a translated image fails, try linking the image with the /nonative linker option.

See "Building Shared Images" in the "Building and Running Synergy Language Programs" chapter of *Synergy Language Tools* for information about building shared images on Synergy Language for OpenVMS.

File structures supported by Synergy Language

Within the OpenVMS environment, file and record processing is controlled by the Record Management Services (RMS). Synergy Language for OpenVMS supports both RMS file organization and Synergy Language stream files.

RMS file organization

The RMS system is a set of services that provides an interface between OpenVMS users and their data. RMS manages the placement and retrieval of records within files, where a record is a logical collection of data treated as a single unit. How records are collected within a given file is determined by the file's organization. There are three basic types of file organization for RMS files:

- Sequential files
- Relative files
- Indexed Sequential Access Method (ISAM) files

In *sequential RMS files*, each record in the file is placed after the record that precedes it. Records can generally only be retrieved from a sequential file in the order that they were written to the file.

In *relative files*, records are stored and retrieved by referencing their relative record numbers within the file. The primary purpose of relative files within RMS is to provide efficient random accessing of records with locking.

Indexed RMS files, or *ISAM files*, have records stored and retrieved according to one or more keys within the record. Records with duplicate keys are always inserted at the end of the sequence of duplicated keys.

See your *OpenVMS Record Management Services Reference Manual* for information on how to use RMS files.

Synergy DBMS stream files

Synergy Language stream files are RMS stream files with carriage control set to CR/LF and for which Synergy Language provides random-access and locking mechanisms. The advantage of stream files is that they enable you to randomly access records using a record number.

Synergy Language stream files have an internal buffer composed of 512-byte blocks. The default buffer size is 8192 bytes. You can modify this size with the BUFSIZ qualifier on the OPEN statement. As the program performs various I/O requests, file blocks containing the requested data are shuffled to and from mass storage as required. The buffer size is also the area locked.

RMS provides locking on a record-by-record basis for relative and indexed files. When a Synergy Language program opens a relative or indexed file in update mode on OpenVMS, RMS record locking is used. Synergy Language locks stream files independently of RMS using the OpenVMS-supported Lock Manager. When a record is read from a stream file that has been opened in update mode, all blocks spanned by that record are locked. If that record is modified, the blocks spanned by that record are flushed to disk to ensure that the modifications are available to other users.

If stream files are used, you must have the SYSLCK privilege in order for the locks obtained through the Lock Manager to be “seen” by other users. Using the SYSLCK privilege does not mean you can lock system-level processes; it merely means that locks can be seen on a system-wide basis.

Installing Multiple Versions of Synergy Language

You can install multiple versions of Synergy Language on the same machine by installing Synergy Language into directories other than [SYNERGYDE.DBL]. The primary version of Synergy Language, enabled on a system-wide basis at startup time, will still be in [SYNERGYDE.DBL].

To install Synergy Language to coexist with a previous version of Synergy Language,

- Use the alternative installation method described in the installation instructions.
- Define SYS$MESSAGE as a search list to include the directory containing **DBLMF.EXE** as well as SYS$COMMON:[SYSMSG].
- Make sure that **SYS$MESSAGE:DBLMF.EXE** is not installed.

Failure to perform these steps (which ensure that the new message file is used) will cause access violations. If required, the new **DBLMF** message file can replace the earlier version in SYS$COMMON:[SYSMSG]. **DBLMF** is backwards compatible.

The alternative installation procedure also enables the system manager to associate the version of Synergy Language being installed with other, perhaps alternative, installations of UI Toolkit, Repository, and ReportWriter. During an alternative installation, the procedure will prompt for the locations of each of these. These will default to the standard locations. **ACTIVATE_SDE.COM** will set up appropriate logicals to enable the use of UI Toolkit, Repository, and ReportWriter.

The installation program asks you for a command. Whatever the installation type, the procedure edits **DIBOL.CLD** to insert the requested command as the verb to invoke the compiler.

Please read your installation instructions thoroughly.

Using the alternative version

You can use the alternative version of Synergy Language on a process-wide basis by invoking **ACTIVATE_SDE.COM** from the appropriate Synergy Language directory. This command file, generated at installation time, sets up appropriate logicals in the process table, installs the command definition file generated at installation time into the process command table, and directs the process to use the new error message file. To activate Synergy Language version *nnn*, use the following:

```
@[dbldirectory_nnn]ACTIVATE_DBL
```

To deactivate the alternative version, and revert to the system-wide installation, invoke **ACTIVATE_SDE** with a nonblank parameter. For example:

```
@[dbldirectory_nnn]ACTIVATE_DBL 1
```

Limitations on OpenVMS

The following sections describe known problems and implementation issues on OpenVMS.

CTRL+C and CTRL+Y trapping

If CTRL+C and CTRL+Y trapping is not enabled using the FLAGS subroutine, typing CTRL+C or CTRL+Y causes an immediate program exit, even from an I/O statement. This OpenVMS limitation is caused by the way AST routines handle CTRL+C and CTRL+Y.

Shareable DECC runtime components

For performance reasons, the Synergy Language install and start-up procedures ensure that the **CMA$TIS_SHR.EXE** DECC runtime component is installed as shareable. SHRIMGMSG in SYS$MESSAGE is also accessed.

8

Statements, Subroutines, and Functions on OpenVMS

Synergy Language Statements

The following sections describe features that are specific to Synergy Language statements on OpenVMS. These statements are described in further detail in the "Synergy Language Statements" and "Defining Data" chapters of the *Synergy Language Reference Manual*.

Creating and opening relative files

If you're creating a relative file, the RECSIZ I/O qualifier is required. If you're opening a relative file for input, the RECSIZ qualifier is not required, but is checked if present, and an "Invalid record size" error ($ERR_IRCSIZ) occurs if the value does not match the maximum record size defined for the file. *Record_size* is the record size for the file being opened. The values 0 and -1 are not valid.

DISPLAY

The DISPLAY statement is valid only on channels opened to a terminal, Synergy stream or sequential files opened in output or append mode, and print files opened in output or append mode.

^EOF

OpenVMS does not support the ^EOF qualifier on WRITE statements.

GET, GETS, PUT, PUTS

The GET, GETS, PUT, and PUTS binary I/O statements are only available on channels opened to Synergy stream files or character-oriented devices such as terminals. They are not available on channels opened to any other file type.

KEYNUM and *key_spec*

If you specify both the RFA and KEYNUM qualifiers on the FIND or READ statement, KEYNUM must specify the primary key as the key of reference (Q_PRIMARY or 0).

If you want to FIND a segmented key that is specified by *key_spec*, you must first construct that key by concatenating each segment together. You can use the %KEYVAL intrinsic function to return the extracted key value from the specified record.

^LAST

OpenVMS supports the ^LAST qualifier on indexed (ISAM) and relative files. ^LAST is not allowed on the WRITE statement.

LPQUE

If your program is interactive and you set system option #22, the LPQUE statement will spawn a PRINT statement to print a specified file, which enables you to add print options after the filename. If, however, your program is not running from an interactive session or option #22 is not set, the LPQUE statement uses the $SNDJBC system service to print a specified file. Any switches that follow the filename are ignored. You cannot use wildcard characters in the file specification.

If the ALIGN option of LPQUE is specified, OpenVMS generates a /hold qualifier on the PRINT command, which causes the current job to be put in a hold queue. If the LPNUM option of LPQUE is specified, the resulting queue name must be defined within the running system. If LPNUM is not specified, the default queue is SYS$PRINT. If LPNUM is specified as an alpha expression, that expression is used as the queue name. If LPNUM is specified as a numeric expression, the queue name is DBL$LP*n*, where *n* is the specified number. See LPQUE in the "Synergy Language Statements" chapter of your *Synergy Language Reference Manual* for the statement syntax and additional information.

O:P mode

If you need to edit files created in O:P mode, set system option #38 before running the program that creates the files. If system option #38 is not set, the files will not look "correct" when you edit them. Additionally, stream files (files opened in **O:S** mode with RECTYPE of 4, 5, or 6) will not look correct when PRINTed or TYPEd. Use **O:P** mode for files to be PRINTed or TYPEd. Also, if you use the FLAGS subroutine (with flag 6 set) to disable carriage control, do not try to PRINT or TYPE the file unless you use O:P mode and set system option #38.

OPEN

OPEN mode O (output) defaults to sequential file type. Use the /stream compiler option to change the default to stream. This is the default for UI Toolkit applications using U_OPEN.

Because Synergy Language defaults to sequential submode, the GET, GETS, PUT, and PUTS statements are not supported on files opened in output mode unless you do one of the following:

- Specify the /stream compiler option when compiling your program.
- Use the /stream qualifier in the OPEN OPTIONS string.
- Specify RECTYPE:4 (or /rectype=4) OPEN qualifier.

The runtime now treats an OPEN of "NL:" the same as "NLA0:"

OPEN statement qualifiers

The following OPEN statement qualifiers are available only on OpenVMS:

- BKTSIZ
- BLKSIZ
- BUFNUM

- BUFSIZ
- CONTIG
- DEQ
- RECTYPE

The following options on the OPTIONS qualifier are meaningful only on OpenVMS:

- /alloc
- /bufnum
- /bufsiz
- /deq
- /rectype

PURGE

If you purge a channel open to a mailbox, the runtime will no longer write an EOF to the mailbox. If you CLOSE a mailbox, it will still write EOF to the mailbox (for compatibility with DIBOL).

Q_EOF and Q_LAST

The qualifier POSITION:Q_EOF is the same as ^EOF. The qualifier POSITION:Q_LAST is the same as ^LAST. See ^EOF and ^LAST on page 8-2.

READ

If you try to READ a record that does not exist in a relative file, the runtime resets the current context to 0 (or "No context"). The context after such an error is undefined and could differ across platforms. Reading a record by RFA on an explicit key of reference other than 0 is not allowed.

REVERSE

The DIRECTION:Q_REVERSE qualifier and the REVERSE keyword are supported on both indexed (ISAM) and relative file types.

SEND

The maximum message length of the SEND statement is 16383 bytes.

STOP

To use the STOP statement to chain to a DCL command, precede the command with a dollar sign as follows:

```
stop "$command [arg1] [arg2] ..."
```

System option #35

If system option #35 is set, the FORMS(*chn, #*) statement on a channel opened to a file with carriage return control will output an extra line feed (for compatibility with VAX DIBOL). If system option #35 is not set, the FORMS statement will only write the number of line feeds to the file necessary to advance the paper the correct number of spaces specified in the FORMS statement.

TEMPFILE

The *temp_spec* argument on the TEMPFILE qualifier is ignored because a new version is used for a new temporary file of the same name. Be careful of directory version limits when using the TEMPFILE qualifier.

TT:

If you open TT:, and TT: is redirected to a disk file, each DISPLAY statement creates a separate record in the file.

When option #39 (or #35) is not set, defining TI or KB as "TT:" is not functionally equivalent to using TT:. Specifying TT: uses SYS$INPUT and SYS$OUTPUT. Defining TI or KB as "TT:" causes the translation of TT: to be used, which usually is defined as the physical terminal device. This is the same as opening TT: when option #39 is set. (Remember that option #35 also sets option #39.)

Synergy Language Subroutines and Functions

OpenVMS-specific subroutines

The following Synergy Language subroutines are available only on OpenVMS. Refer to the "System-Supplied Subroutines and Functions" chapter of the *Synergy Language Reference Manual* for descriptions of these subroutines.

ASTRST – Restore the contents of work areas used as the result of an AST

ASTSAV – Save the contents of work areas used as the result of an AST

CREMBX – Create a mailbox

DBL$DEVCLT – Get the class and type of a device

DBL$EXECUTE_IMAGE_ROUTINE – Execute a routine contained in a shareable image

DBL$SETKRF – Set the key of reference for the next operation on an ISAM file

DBL$SNDOPR – Send a message to the system operator

DELMBX – Mark a permanent mailbox for deletion

EMPBUF – Write out modified I/O buffers

ENDFL – Position the file pointer after the last record of a file

%FSTAT – Return the value of the last floating point call

FXSUBR – Dispatch to a floating-point function

GETCM – Get data from the process message area

PURGE – Delete previous versions of a file

PUTCM – Store data in the process message area

SETCTL – Modify the operation of control characters

SORT – Provide a callable interface to DBLSORT

%SUCCESS – Determine if low-order bit is on or off

TT_NAME_TO_NUMBER – Convert an OpenVMS terminal name to its equivalent terminal number

TT_NUMBER_TO_NAME – Convert a terminal number to its equivalent OpenVMS terminal name

TTBRDCST – Enable a program to trap broadcast messages

TTCHAR – Return type, lines, and width of a file

TTFLGS – Set OpenVMS-specific terminal processing options (some available on Windows and UNIX)

TTMBX – Associate a mailbox with a channel opened to a terminal device

VMCMD – Execute a DCL command

VMMSG – Get the text of an OpenVMS system message

^XTRNL – Return the value of a global symbol

In addition, flags 8, 9, and 10 of the DFLAG subroutine are available only on OpenVMS, as are several of the keywords for the GETFA subroutine.

Routines that work differently on OpenVMS

This section describes Synergy Language routines that function differently on OpenVMS than on other operating systems. Refer to the "System-Supplied Subroutines and Functions" chapter of the *Synergy Language Reference Manual* for a full description of these routines.

BTOD and DTOB

In addition to integer data and decimal data, the BTOD and DTOB subroutines also convert quadwords (64 bits) on OpenVMS systems.

CMDLN

The CMDLN subroutine returns the command line in uppercase characters and does not include the full path name of the Synergy runtime or the program name. To use this subroutine on OpenVMS, start the program with a foreign symbol. Refer to your *OpenVMS User's Manual* for information about how to define a symbol as a foreign command.

For example, if we set up a foreign symbol as follows for a program called **MAIN** (where $PATH was previously set as an environment variable pointing to the directory that contains **MAIN.EXE**):

```
prog:==$PATH:MAIN
```

we could use the CMDLN subroutine in this program as follows:

```
main MAIN
.define TTCHN       ,1
record
   buffer           ,a80

proc
   open(TTCHN, o, "tt:")
   xcall cmdln(buffer)
   writes(TTCHN, "Buffer = "+buffer)
endmain
```

If we then ran this program with the following command line arguments:

```
prog arg1 arg2 arg3
```

the output would be as follows:

Buffer = arg1 arg2 arg3

However, if we ran the program MAIN without arguments, or if we ran this program without a foreign symbol command, the output would be the following:

Buffer =

DELET

Not only can you use wildcard characters in the filename specification on the DELET subroutine, but a given file can have more than one version. If the filename specification in DELET does not explicitly specify the version number, all versions of the file will be deleted.

ERROR and %ERROR

Both the ERROR subroutine and the %ERROR function have an optional fourth argument that returns the RMS STV value associated with the last RMS system call if it exists. This argument will *not* cause an error on UNIX and Windows; it will simply return a 0.

EXEC

The EXEC subroutine uses the DECC **execvp** function call. You must set the logical VAXC$PATH to a search list where the program image will be found. EXEC is only available on OpenVMS and UNIX.

FATAL

The FATAL subroutine uses the translation of the DBL$FATAL_IMAGE logical instead of *filename* when system option #3 is specified and as the default program to chain to.

FLAGS

Flag 3 of the FLAGS subroutine works differently with the OPEN statement and RENAM subroutine on OpenVMS than it does on other systems. Normally, flag 3 protects the runtime from accidentally overwriting a file by generating a “Cannot supersede existing file” error ($ERR_REPLAC). To generate this error on OpenVMS, you must also specify the output filename’s version number in RENAM or OPEN. System option #35 will cause this error to be generated in RENAM if any version of the target file exists.

The default value for flag 4 is the current terminal setting.

Flag 6 changes a file that is open for output to have no record attributes. It also disables carriage control on **O** mode files (except stream files).

GLINE

The GLINE subroutine uses the OpenVMS LIB$GET_INPUT subroutine to get an input line from the device assigned to the logical name SYS$INPUT.

ISAMC

You can use the ISAMC subroutine to create RMS ISAM files. The PAGE, ALLOC, MULTIPLE, and STATIC_RFA options in the filename specification are ignored on OpenVMS. Also, records that contain duplicate keys are always inserted at the end of the list of duplicates, which means that the NOATEND option in the key specification will generate “Illegal key specified” error ($ERR_BADKEY).

You can specify the following COMPRESS options for RMS ISAM:

[NO]INDEX	Compresses the index. The default is INDEX.
[NO]KEY	Compresses the key within the data. The default is KEY.
[NO]RECORD	Compresses the record within the data. The default is RECORD.
ALL	All of the above.

See the ISAMC subroutine in the “System-Supplied Subroutines and Functions” chapter of the *Synergy Language Reference Manual* for more information.

For better performance, you should not create files with ISAMC; you should use the FDL editor and open files in O:I mode with the FDL qualifier.

ISSTS

Positions 22 through 29 of the status argument in the ISSTS subroutine will be filled with “09999999”.

JBNO

The JBNO subroutine optionally returns the process identification number, the owner PID, and the group identification number. Any specified ID number should be at least 10 digits wide to prevent overflow.

KILL

The KILL subroutine terminates the calling process by making a call to the $DELPRC system service. KILL is equivalent to executing a STOP statement followed by the LOGOUT command. If the calling Synergy program is running as a subprocess (created by the RUNJB or VMCMD subroutine), that subprocess will be terminated and control will return to the parent process.

OPENELB

Synergy Language on OpenVMS uses shared executable images to implement ELBs. The OPENELB subroutine adds the referenced shared images to the active list of shared images for subsequent access through calls to the XSUBR subroutine. OPENELB performs no other action in the OpenVMS environment.

PARSE

Several of the arguments to the PARSE subroutine are specific to OpenVMS: *node*, *device*, and *version*. On OpenVMS, PARSE uses the RMS $PARSE facility.

POSRFA

The POSRFA subroutine does not support a non-0 key-of-reference specification.

RENAM

If the new filename specification doesn't contain an explicit version number on a call to the RENAM subroutine, and a file already exists with the same name, the file is renamed with the next higher version number. If the new filename specification does contain an explicit, non-0 version number, and a file of the same name and version number already exists, RENAM will replace the existing file (unless flag 3 is set on the FLAGS subroutine). Using system option #35 with RENAM causes all target files to be deleted before the rename is performed, unless flag 3 is set. Then, if any target files exist, a "Cannot supersede existing file" error ($ERR_REPLAC) will be generated.

RUNJB

The RUNJB subroutine works differently on different operating systems. For details, see the RUNJB subroutine in the "System-Supplied Subroutines and Functions" chapter of the *Synergy Language Reference Manual*.

SETDFN

The initial default file specifications used by the runtime are different on OpenVMS. For details, see the SETDFN subroutine in the "System-Supplied Subroutines and Functions" chapter of the *Synergy Language Reference Manual*.

SETLOG

If you don't pass a *translation* value, SETLOG will delete the specified logical. You can define a search list logical by separating the elements of the list with commas. To preserve commas in the *translation* value, use quotation marks. To preserve quotation marks, use two consecutive quotation marks.

SHELL

The SHELL subroutine works differently on each operating system. For details, see the SHELL subroutine in the "System-Supplied Subroutines and Functions" chapter of the *Synergy Language Reference Manual*.

SPAWN

The SPAWN subroutine executes a DCL command string and sends the command string to the Command Language Interpreter (CLI) as input. SPAWN should be used sparingly on OpenVMS due to its high CPU overhead.

TFLSH

The TFLSH subroutine is only available on OpenVMS and UNIX. On OpenVMS, it ensures that the previous asynchronous terminal I/O to the controlling terminal has finished.

TNMBR

The TNMBR subroutine will use the physical device to determine the terminal number for a virtual terminal if system option #17 is set. If this option is not set, TNMBR will use the VT device specification to determine the terminal number. See the TNMBR subroutine in the "System-Supplied Subroutines and Functions" chapter of the *Synergy Language Reference Manual* for more information about how terminal numbers are assigned by TNMBR. Also see "Terminal Numbers" on page 9-3 of this manual for more information.

TTNAME

On OpenVMS, if the program is running as a batch job, the TTNAME subroutine returns the null device specification: "_NLA0:".

WAIT and TTSTS

If you are running a detached job and you specify position 2 or 3 in the *parameters* argument of the WAIT subroutine, Synergy Language will return position 2 in the *event* argument (which indicates that input is pending). TTSTS will return a value of 1 in its *status* argument (which also indicates that input is pending).

These same values are returned from subroutines WAIT and TTSTS if you redirect SYS$INPUT from a command file and system option #39 is set. (Option #39 sets this behavior; option #35 only sets #39.)

XSUBR

The XSUBR subroutine does not have direct access to any shared images linked to the program. You must call OPENELB to access routines in a shared image. If the target routine is not found in the main program image, LIB$FIND_IMAGE_SYMBOL is used to search the shared image list created by calls to OPENELB. Only Synergy Language routines can be invoked by XSUBR.

Subroutines that have no meaning on OpenVMS

The following subroutines perform no function on OpenVMS and will generate an error if called:

- BREAK
- FORK
- INITPORT
- LM_KCR
- LM_LOGIN
- LM_LOGOUT
- SERIAL
- STTY
- W_CAPTION

AST support in Synergy Language

Synergy Language fully supports AST routines written in any of the re-entrant HP OpenVMS languages. These AST routines must comply with the limitations of the language in which they are written and with the limitations of OpenVMS in regard to asynchronous processing.

A number of obstacles prevent unhindered use of Synergy Language at the AST level. Although the Synergy Language implementation allows for external asynchronous processing at the language statement level, the non-re-entrancy of some statement processors prevent it from being allowed at the system level. What this means is that the Synergy programmer is allowed to do an implied XCALL or function reference “underneath” a Synergy program *between* Synergy Language statements, but not *during* some of those statements. Since OpenVMS generates ASTs asynchronously at the system level, some Synergy Language statements cannot be supported at the AST level.

The reason for this is that the Synergy runtime makes extensive use (from the C standpoint) of “static” data. A Synergy Language routine running at the AST level could corrupt the static data areas used by the currently active Synergy Language statement that’s being processed at the non-AST level.

Another problem is that the Synergy runtime is implemented in C, and certain C runtime functions are not re-entrant for similar reasons.

The following operations should not be performed in Synergy Language AST routines:

- I/O on any channels that may be in use at the non-AST level
- Message sending and receiving
- SLEEP statements

By noting the above limitations and using the ASTSAV and ASTRST subroutines, AST service subroutines can be implemented in Synergy Language.

DBLSTARLET directory

A subset of the STARLET library, called DBLSTARLET, is available in Synergy Language for OpenVMS. We also provide a program called **CONVERTER** in the DBLSTARLET directory. **CONVERTER** enables you to extract any additional modules from STARLET that your applications requires.

If you use STARLET offset values as array indices, you must add one to these values; the offsets are 0-based and Synergy Language subscripts are one-based.

Floating-point arguments

Any routine that passes floating-point arguments by value will not work on either the Alpha or the I64. This type of routine requires a C wrapper routine.

Record Locking

If you're migrating from UNIX, be aware that record locks on OpenVMS are channel-based. If the same program opens the same file on two different channels in update mode, both channels will be affected by each other's locks, which may cause unexpected $ERR_LOCKED errors.

9

Other OpenVMS-Specific Information

ISAM Utilities

The following ISAM utilities either work differently or are not available on OpenVMS:

- The **status** utility always returns 90000000 as the number of records in an RMS ISAM file. There is no way to find the number of records in an RMS ISAM file unless you read sequentially through the file.
- The **ipar** utility, which generates parameter file descriptions of existing ISAM files, is not available on OpenVMS. Use the ANALYZE/RMS utility to extract file descriptions and check file integrity. Refer to your *Record Management Utilities Reference Manual* for more information.
- The **irecovr** utility, which converts ISAM files from their previous Synergy Language version, is not available on OpenVMS.
- The **ismvfy** utility, which verifies several aspects of a ISAM file's structure, is not available on OpenVMS.

Terminal Numbers

Synergy Language uses terminal numbers in the TNMBR subroutine and in one form of the SEND statement. Terminal numbers are determined in one of the following ways:

- Synergy Language uses the environment variable TNMBR in the current process if it is set. With this method, you can easily give the same terminal number to more than one terminal. For example, if two people define TNMBR equal to 1 in their login file and both are logged in at the same time, both of their terminals will have the number 1.
- The TNMBR subroutine returns one of the following values and assigns the corresponding number:

0	OPA0:
-1	The job is running detached, regardless of the terminal device specification.
-2	The process is a network process.
-3	The process is a batch process.
number	All other instances

  On OpenVMS, Synergy Language calculates a unique number for the device name in the form TT*cn*, where *c* is a controller letter and *n* is a unit number. The device type (for example, TT) may vary, depending on the type of terminal controller used. Terminal numbers are not necessarily compatible with other platforms or with DIBOL terminal numbers, and they may vary from release to release. We recommend that you make no assumptions as to the correspondence between terminal numbers and terminal names, other than the uniqueness of a terminal number for a local system.
- For a virtual terminal, the TNMBR subroutine will use the physical device to determine the terminal number if system option #17 is set. If this option is not set, TNMBR will use the terminal device name of the virtual terminal to determine the terminal number.

If you want to know the terminal number for a particular device, you can use the TT_NAME_TO_NUMBER subroutine to convert an OpenVMS terminal name to its equivalent terminal number. A terminal number can be up to eight digits long.

Peripheral Devices

Printer setup

You have the following options when printing on OpenVMS:

- For spooled printers, you can open the device directly. OpenVMS directs the output through the spooling system so the printer is not locked by one process.
- You can use the LPQUE statement to add a job to the print-batch services queues. If you use LPQUE, the default condition is to use the $SNDJBC system service, which speeds up spooling. The drawback to this is that you can't use wildcard characters in the file specification, and print switches on filename specifications are ignored.

  LPQUE can also be used to submit batch jobs if its queue is a batch queue.

  However, if you specify system option #22, Synergy Language constructs a DCL PRINT command line and spawns a subprocess to execute the PRINT command line. With this option, you can use wildcard characters, and valid print switches on a filename will be sent through to subprocesses as part of the PRINT command line.

  If system option #22 is set, printing errors may not be reported to your program.
- For nonspooled printers, you can open the device directly. It is the program's responsibility to handle device sharing between processes.
- If you set DFLAG flag 8, the additional parameter SJC$_NOPAGINATE is used, which is equivalent to using a PRINT/NOFEED DCL command.

Synergy Language and LTA devices

Synergy Language does not automatically connect to an outbound LTA device, such as a modem or printer connected to a terminal server (neither do other OpenVMS languages including DIBOL). This is due to the varied and specific nature of such requests, which are dependent on the version of LAT in OpenVMS and the firmware in the terminal server.

If you need to perform I/O to an LTA device, you must perform an "LAT connect QIO" to the device after the OPEN statement and an "LAT disconnect QIO" before the CLOSE. We have included an example routine, LAT, in the file **LAT.DBL** in the DBLSTARLET directory. The LAT subroutine shows the code required to negotiate different connections and the types of timing required depending on which device is connected to the terminal server at the other end. For more information on the code used in **LAT.DBL**, we suggest you read the *OpenVMS I/O User's Reference Manual* in the OpenVMS documentation set, or call HP for an explanation of LAT-specific QIO mechanisms.

When using modems we suggest that you only use the latest firmware on all Compaq terminal servers. For DS300, DS700, and DS90TL, we recommend a minimum of BL45C-14; for DS200, we recommend version 3.3. Using the latest firmware will solve many potential timing and flow control problems. If you are using an OpenVMS version lower than 6.1, we suggest you contact your HP support center for up-to-date LAT patches to fix problems in the LTA and LATACP drivers that could affect your ability to debug. On Alpha 6.1, these patches also fix random protocol disconnect errors for logged-in users. (The **LAT.DBL** program assumes that these patches are installed.)

To help test LAT connections, we provide a **LATT.DBL** program which is an example of how to use the LAT subroutine. You can run this program to test the results for an outbound LTA device on your system.

We also include a **LATMSGDEF.DBL** file in DBLSTARLET, which was provided by Compaq as an aid in documenting possible LTA device error codes besides the 19 documented in **LATT.DBL**.

There are potential problems with using a LAT disconnect with a channel whose output has not yet been flushed. In some cases, the program will hang in an LEF state forever. You can alleviate this potential problem by calling the %TTSTS function or the TFLSH subroutine on the channel to ensure that the runtime's asynchronous QIO has completed before issuing the disconnect.

System Options

The following system options are specific to or function differently on OpenVMS. Refer to the "System Options" chapter of *Environment Variables and System Options* for more information about each system option.

- When you define multiple system options with DBLOPT on OpenVMS, make sure you enclose the options in quotation marks. For example:

  ```
  define DBLOPT "1,7,16,35"
  ```

  If you don't use quotation marks, the runtime will only process the first option specified.
- System option #7, which determines whether the runtime will use the Synergy message manager or the local message facilities, is on by default. Use system option #47 to disable use of the Synergy message manager.
- System option #16, which maps the quit character to the interrupt character, is only available on OpenVMS and UNIX. On OpenVMS, you can restart the program at the point of interruption by issuing a DCL CONTINUE command if option #16 is not set.
- If you set option #17 on OpenVMS, the TNMBR subroutine will use the physical device to determine the terminal number for a virtual terminal.
- System option #18, which controls how the in-place MERGE handles the logical end-of-file, is not available on OpenVMS.
- If you set option #22 and the program is running from an interactive session, the LPQUE statement will spawn a PRINT statement to print a specified file, which enables you to add PRINT options after the filename.
- System option #23, which determines where the in-place MERGE places duplicate records, is not available on OpenVMS.
- Some of the VAX DIBOL-compatible functionality provided by system option #35 is only available on OpenVMS.

Message Facilities

System option #7, which enables you to use the Synergy message manager, is set by default on OpenVMS. If you set option #47, Synergy Language will use the local message facility instead of the Synergy message manager for SEND and RECV statements.

The maximum message ID length on OpenVMS systems is 39 characters.

Starting the message manager

You can start (or restart or kill) the message manager with the command

```
$@DBLDIR:dblmsgctlstartup [option]
```

option

(optional) One of the following options:

START	(default) Start the message manager.
RESTART	Restart the message manager.
KILL	Kill the message manager.

The message manager is started automatically in the **SYNERGY_STARTUP.COM** file.

Error Handling

- Synergy Language provides an exit handler on OpenVMS: LIB$SIGNAL and LIB$STOP issue program tracebacks when called from Synergy programs.

You are responsible for setting a flag to make sure your exit handler is not re-entrant. If you don't, an endless loop may occur if you get a Synergy Language error in your handler. The Synergy debugger does not debug exit handlers, and a Synergy Language exit handler may be invalid if the runtime exits abnormally. C and MACRO are the best languages for exit handlers.

- When using LIB$SIGNAL, if you specify an error with a severity of success, warning, or informational, the exception handler will issue the error message and continue processing after the LIB$SIGNAL call. If you specify an error with a severity of fatal or error, or if you're using LIB$STOP, the exception handler will issue a fatal "Unexpected VMS system error" (VMSERROR), followed by the signalled error and the Synergy Language traceback.

LIB$STOP will cause a fatal exit regardless of the severity of the error you specify.

- If the compiler encounters a fatal system error when trying to open or access a file, it will report the associated system error text.
- If an internal, untrapped, unexpected OpenVMS/RMS error occurs in a Synergy Language program, it will be loaded into the DCL $status symbol on exit from the program. The actual system error (not the VMSERROR number) will be loaded into $status. If any other Synergy Language error occurs, it will be loaded into $status on exit.
- The Synergy runtime uses asynchronous terminal output for better terminal performance. Therefore, Synergy Language will report any I/O error, such as $ERR_DEVOFFLINE, on the next terminal input or output statement when the wait for previous I/O completion occurs.
- Synergy programs may generate a "Failure during I/O operation" error ($ERR_IOFAIL) with the SS$_DATAOVERUN system error code if the type-ahead buffer is filled and the terminal is set "nohostsync." (This is a normal OpenVMS error condition.) You can avoid this error by either trapping errors on your ACCEPT statements or ensuring that the terminal is set "hostsync." If the terminal *cannot* be set "hostsync," you can set the terminal "altype" to reduce the occurrence of this error.

Interface with Other Languages

You can link any object module with a Synergy program and call it directly from your Synergy Language code. You can also call Synergy Language routines from non–Synergy Language main routines, and you can call any OpenVMS library or system function from your Synergy Language code.

Synergy Language's C interface supports various C string and data conversion functions for interfacing with C language modules. Refer to the file **xcallv.h** for descriptions of the supported functions.

10

Porting Between OpenVMS and Windows or UNIX Systems

Porting OpenVMS Code to Windows and UNIX

Keep the following in mind as you port OpenVMS code to Windows and UNIX:

- The TTFLGS subroutine only supports flag 4.
- The PURGE subroutine is ignored.
- You must specify the RECSIZ qualifier when you open a relative file containing integer data.
- On OpenVMS, if you don't specify a record size when you open a relative file, the runtime determines the record size by looking at the file being opened. If you specify the record size on the OPEN and it is different from the actual record size in the file, an error is generated.
- On Windows and UNIX, if you don't specify the record size on the OPEN, the runtime opens the file with no error. The record size is determined by the first READ from the file. If you pass a record buffer of a different size than that of the record in the file, an error is generated on the READ.
- The DBL$PARSE subroutine does not add default extensions; it just parses an existing file specification as passed.
- The TTSTS subroutine only returns 1 or 0, not the number of characters in the type-ahead buffer on some UNIX systems. On UNIX, to return the number of characters, TTSTS requires the POSIX FIONREAD ioctl modifier support in the operating system.
- SEND/RECV names are limited to six characters (instead of 39), and SEND/RECV works differently when ported to Windows and UNIX. The default maximum message size is 4096, which can be configured by using the **-b** option on the **synd** program. On OpenVMS, the maximum message size is 16383. See "Messaging" in the "Welcome to Synergy Language" chapter of the *Synergy Language Reference Manual* for more information on operating system differences when sending and receiving messages.
- Most of system option #35's functionality does not apply on Windows and UNIX.
- The following OpenVMS-specific subroutines and functions are unavailable when porting to Windows and UNIX:

  ASTRST

  ASTSAV

  CREMBX

  DBL$DEVCLT

  DBL$EXECUTE_IMAGE_ROUTINE

  DBL$TTCHAR

  DELMBX

  TTBRDCST

VMCMD

VMMSG

^XTRNL

- You cannot use dollar signs ($) in filenames on UNIX systems.
- Some UNIX systems restrict filenames to eight characters and file extensions to three characters.
- Because there are no file version numbers, "*;nnn*" extensions are not allowed.
- OpenVMS system services cannot be used.
- Any C subroutines must be changed to access arguments as described in the "Synergy Language C Interface" chapter of the *Synergy Language Reference Manual*.
- Shared images become ELBs. You cannot limit common definitions within an ELB to be nonvisible externally. This means that common variables must be unique across all ELBs. Nonunique common variables will cause various and unpredictable runtime results. The linker does not check for uniqueness.
- You cannot overlay unnamed global/external commons with records on Windows and UNIX.
- Synergy Language on Windows and UNIX supports alphanumeric (**a**), decimal (**d**), and integer (**i**) keys on ISAM files.
- The SORT statement does not allow integer keys.
- Flag 4 in the TTFLGS subroutine does not affect the WD_ACCEPT, WD_GETS, and WD_READS options of the W_DISP subroutine. You must program your own interpretation of the escape sequences.

Porting Windows and UNIX Code to OpenVMS

Keep the following in mind as you port Windows and UNIX code to OpenVMS:

- On OpenVMS, the DCL command parser parses command lines. It does not support the following batch file syntax:

  ```
  $dbl
  /refresh/object=objfile srcfile
  /refresh/object=objfile srcfile
  /refresh/object=objfile srcfile
  ```

  You will need to modify your batch files to include "$dbl" at the beginning of each line, as follows:

  ```
  $dbl/refresh/object=objfile srcfile
  $dbl/refresh/object=objfile srcfile
  $dbl/refresh/object=objfile srcfile
  ```

- The SHELL subroutine commands are different on OpenVMS than on Windows and UNIX.
- Do not use the SPAWN subroutine unless it is absolutely necessary; it is very slow on OpenVMS systems.
- Environment symbols become logical names.
- Filenames can have up to 40 characters for each component (filename, device, and directory element), up to a maximum of 254 characters.
- Don't use **isload** to load files or the ISAMC subroutine to create them. Load files with the CONVERT/FAST/NOSORT command, and create them from an FDL file created with the EDIT/FDL command or the OPEN O:I statement.
- To reclaim space occupied by deleted records, you must reorganize files from time to time.
- The overhead of doing single character ACCEPT or DISPLAY statements for I/O is significantly higher.
- Write disk I/O (STORE/DELETE/OPEN/WRITE/WRITES/FORMS) is not cached on OpenVMS systems, and excessive use can cause an application to appear much slower than on UNIX. This performance would be apparent in a multiuser situation on UNIX systems.
- Application start-up time is slower on OpenVMS than UNIX, as OpenVMS pages in the application and its shared images. To counteract this, either use bound programs, or keep your applications in shared images activated with the XSUBR and OPENELB subroutines.
- Files have versions on OpenVMS. Opening a file for output will create a new version of the file and will not overwrite the original unless you always append a version number to the file specification. TEMP files are implemented as new versions of a file.

- When trapping errors on I/O statements, you should also report the number returned from %SYSERR. Doing so will help you understand which OpenVMS system error occurred. The number of errors returned on OpenVMS is much greater than on UNIX systems.
- OpenVMS systems only have global message queues.
- The C interface does not exist as such on OpenVMS systems; the interface from Synergy Language to other languages is defined in OpenVMS's architecture reference manual, available from HP.
- OpenVMS ISAM files use RMS. READ REVERSE only works on ISAM files.
- You must use the RECSIZ OPEN statement qualifier when you open a relative file for output.
- The DETACH statement is not implemented.
- The FORK subroutine is not implemented.

Index

E

F

S